Karlsruhe Institute of Technology

Scalable High-Quality Graph and Hypergraph Partitioning

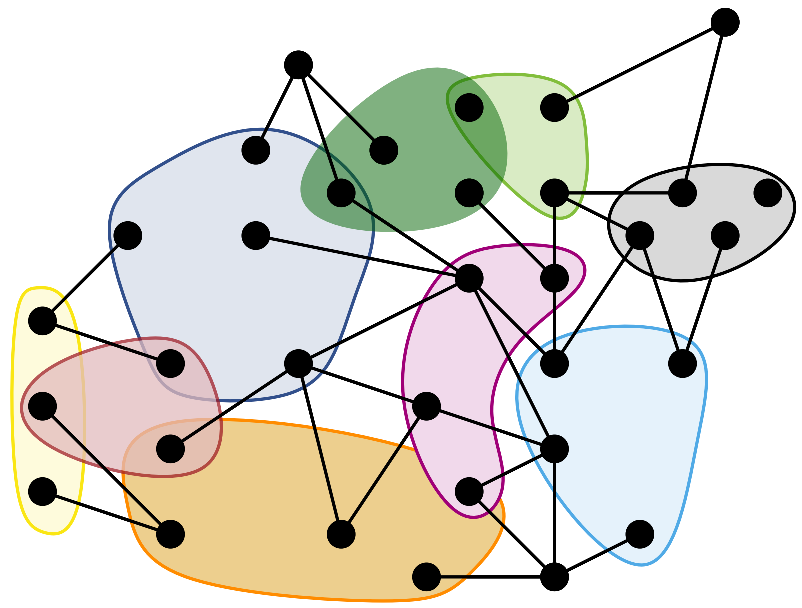
June 13, 2022

Lars Gottesbüren, **Tobias Heuer**, Peter Sanders, Sebastian Schlag

Hypergraphs

Karlsruhe Institute of Technology

- generalization of graphs
  $\Rightarrow$ hyperedges connect $\geq$ 2 nodes
- graphs $\Rightarrow$ dyadic ($\mathbf{2}$-**ary**) relationships
- hypergraphs $\Rightarrow$ ($\mathbf{d}$-**ary**) relationships
- hypergraph $H = (V, E, c, \omega)$
  - vertex set $V = \{1, ..., n\}$
  - edge set $E \subseteq \mathcal{P}(V) \setminus \emptyset$
  - node weights $c : V \to \mathbb{R}_{\geq 1}$
  - edge weights $\omega : E \to \mathbb{R}_{\geq 1}$

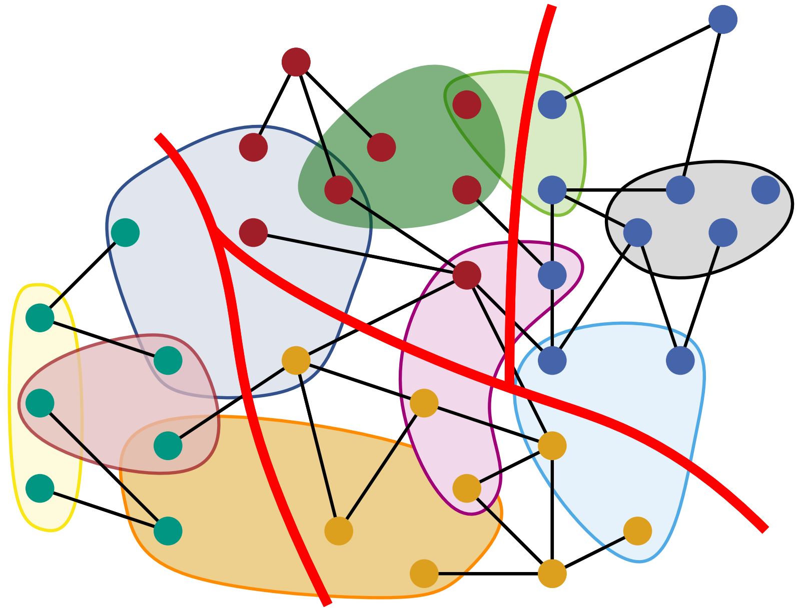


ε-Balanced Hypergraph Partitioning Problem

Partition hypergraph $H = (V, E, c, \omega)$ into $\mathbf{k}$ disjoint blocks $\Pi = \{V_1, \ldots, V_k\}$ such that:

- blocks V_i are **roughly equal-sized**:

$$c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil$$



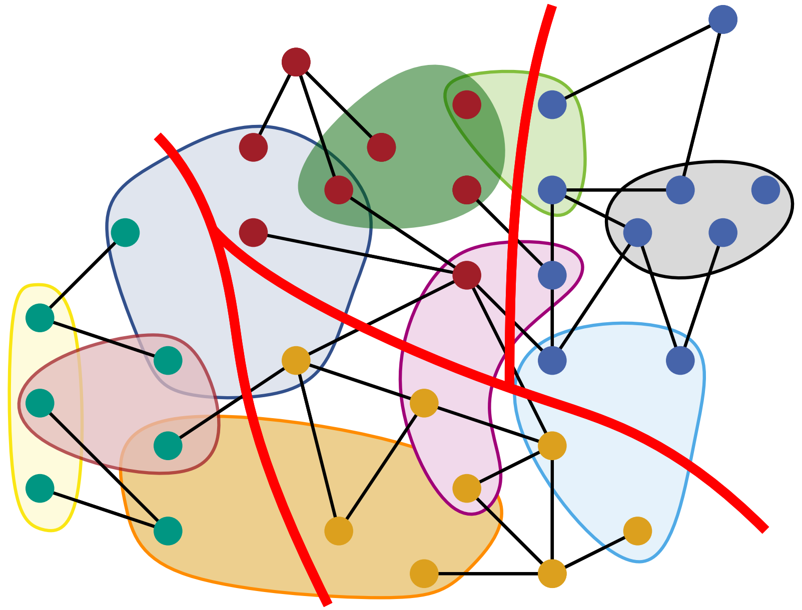
ε-Balanced Hypergraph Partitioning Problem

Partition hypergraph $H = (V, E, c, \omega)$ into $\mathbf{k}$ disjoint blocks $\Pi = \{V_1, \ldots, V_k\}$ such that:

- blocks V_i are **roughly equal-sized**:

$$c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil$$

imbalance parameter



ε-Balanced Hypergraph Partitioning Problem

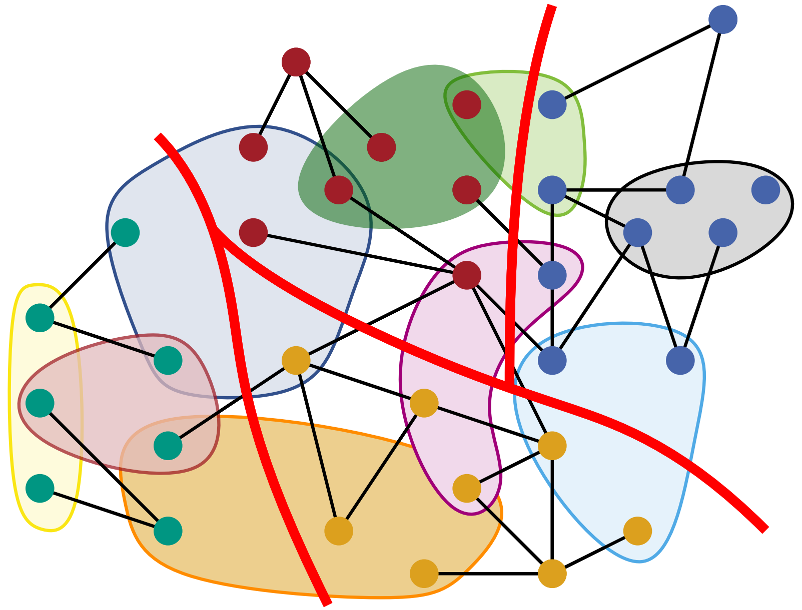
Partition hypergraph $H = (V, E, c, \omega)$ into $\mathbf{k}$ disjoint blocks $\Pi = \{V_1, \dots, V_k\}$ such that:

- blocks V_i are **roughly equal-sized**:

$$c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil$$

imbalance parameter

- **connectivity** objective is **minimized**:



ε-Balanced Hypergraph Partitioning Problem

Karlsruhe Institute of Technology

Partition hypergraph $H = (V, E, c, \omega)$ into $\mathbf{k}$ disjoint blocks $\Pi = \{V_1, \ldots, V_k\}$ such that:

- blocks V_i are **roughly equal-sized**:

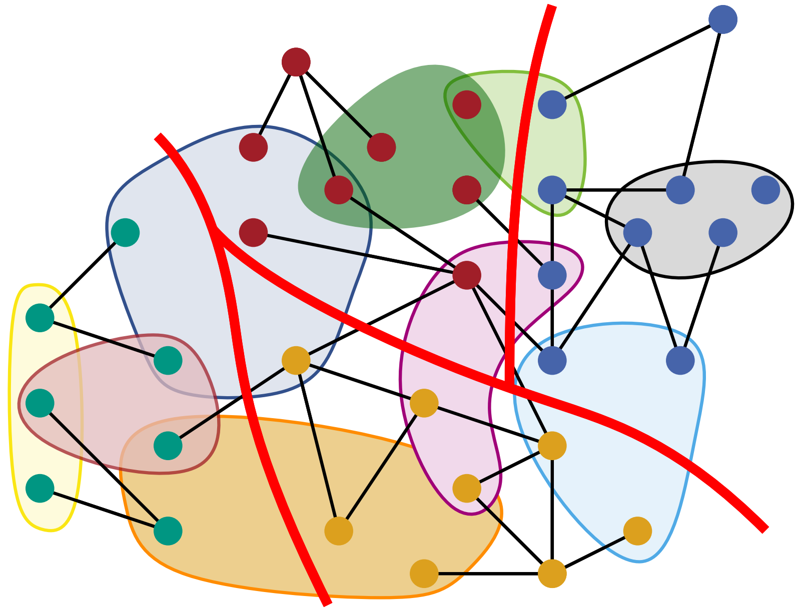
$$c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil$$

imbalance parameter

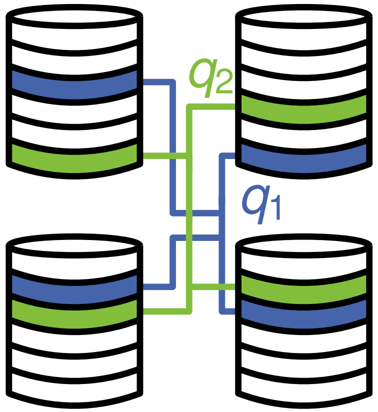
- **connectivity** objective is **minimized**:

$$\sum_{e \in E} (\lambda(e) - 1)\, \omega(e) = 12$$

connectivity
 # blocks connected by net e

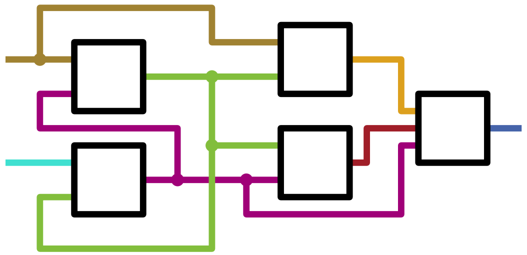


Applications



Distributed Databases

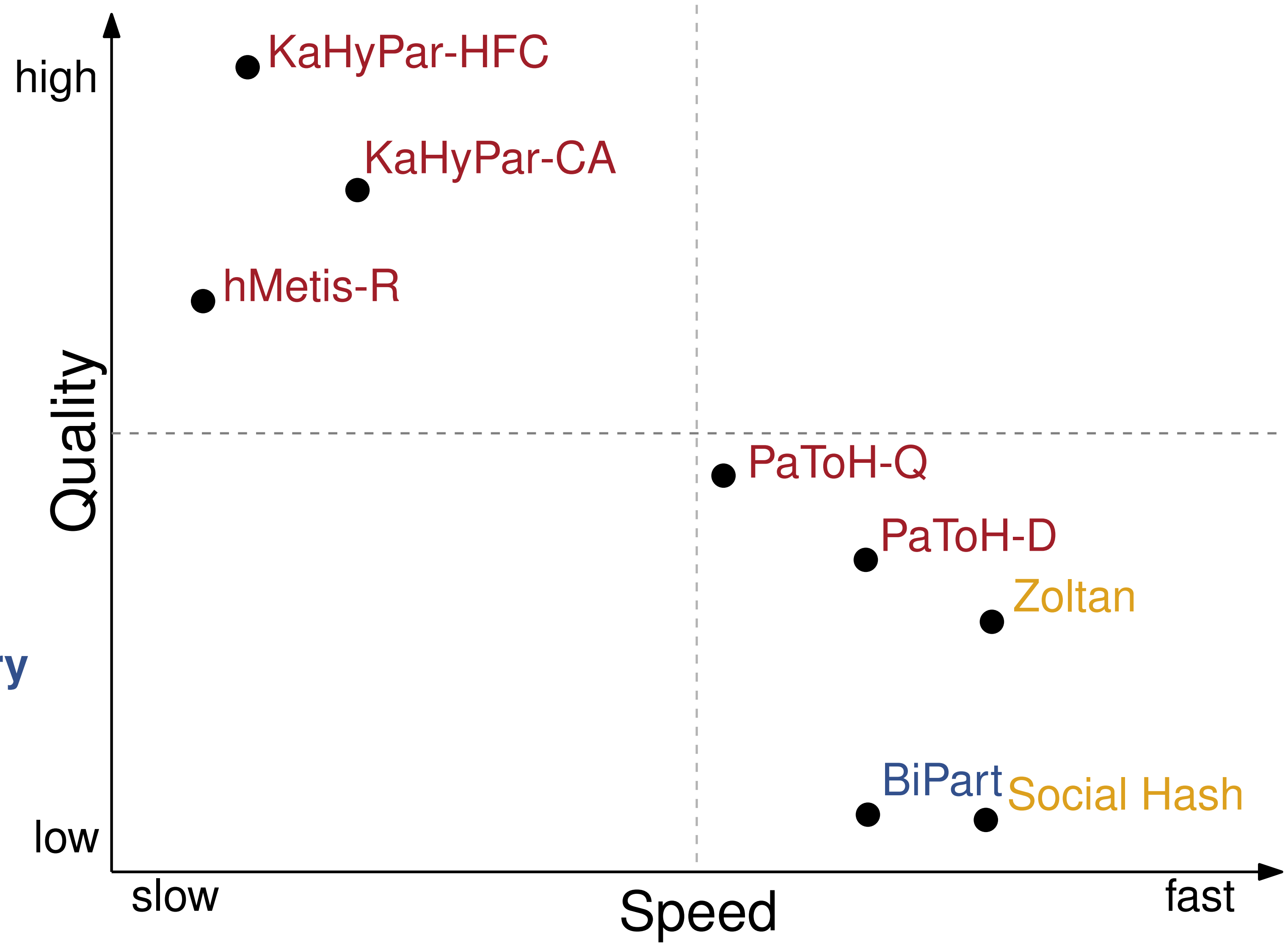
Route Planning



VLSI Design

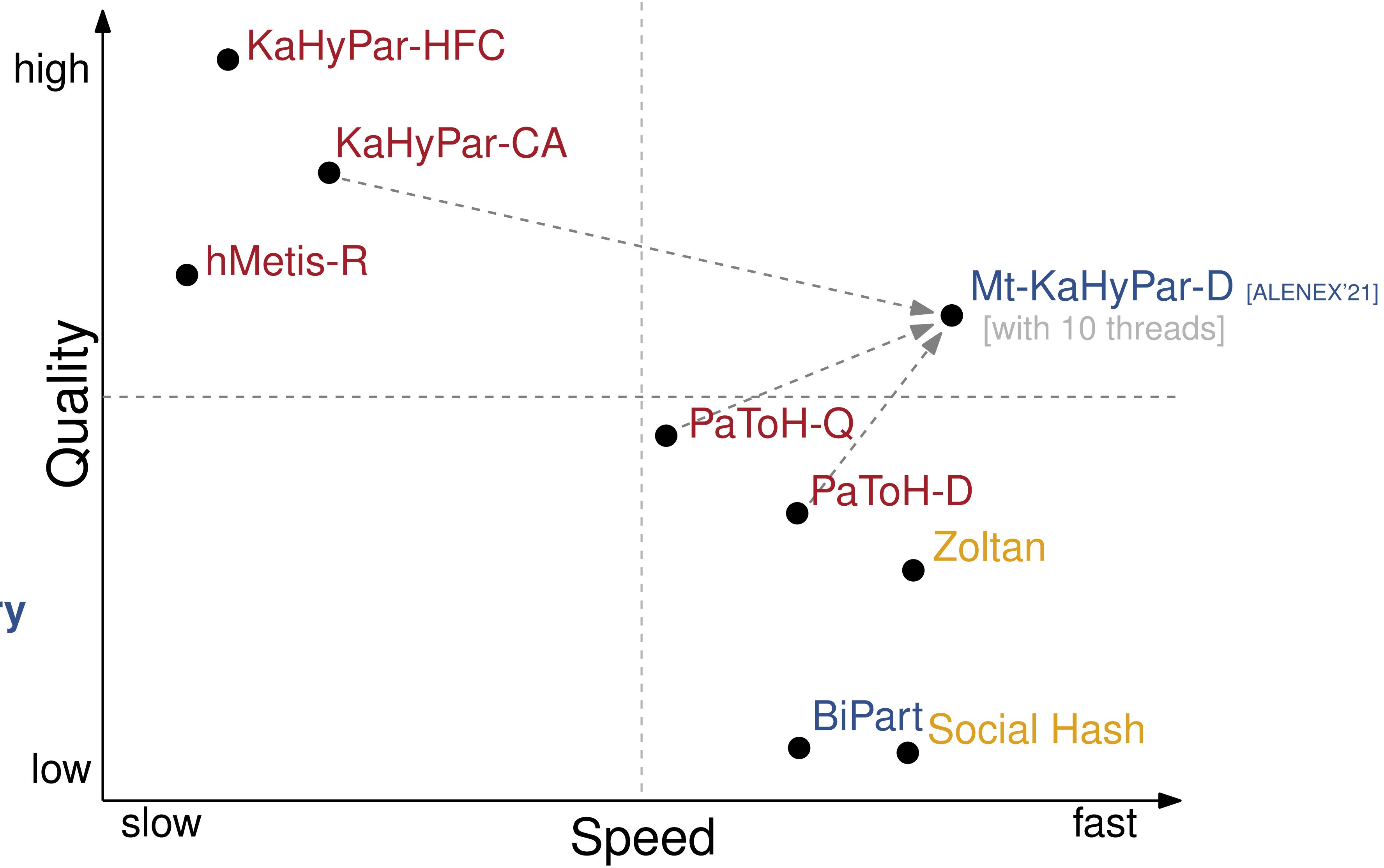
HPC

Trade-Off Landscape for Hypergraph Partitioning



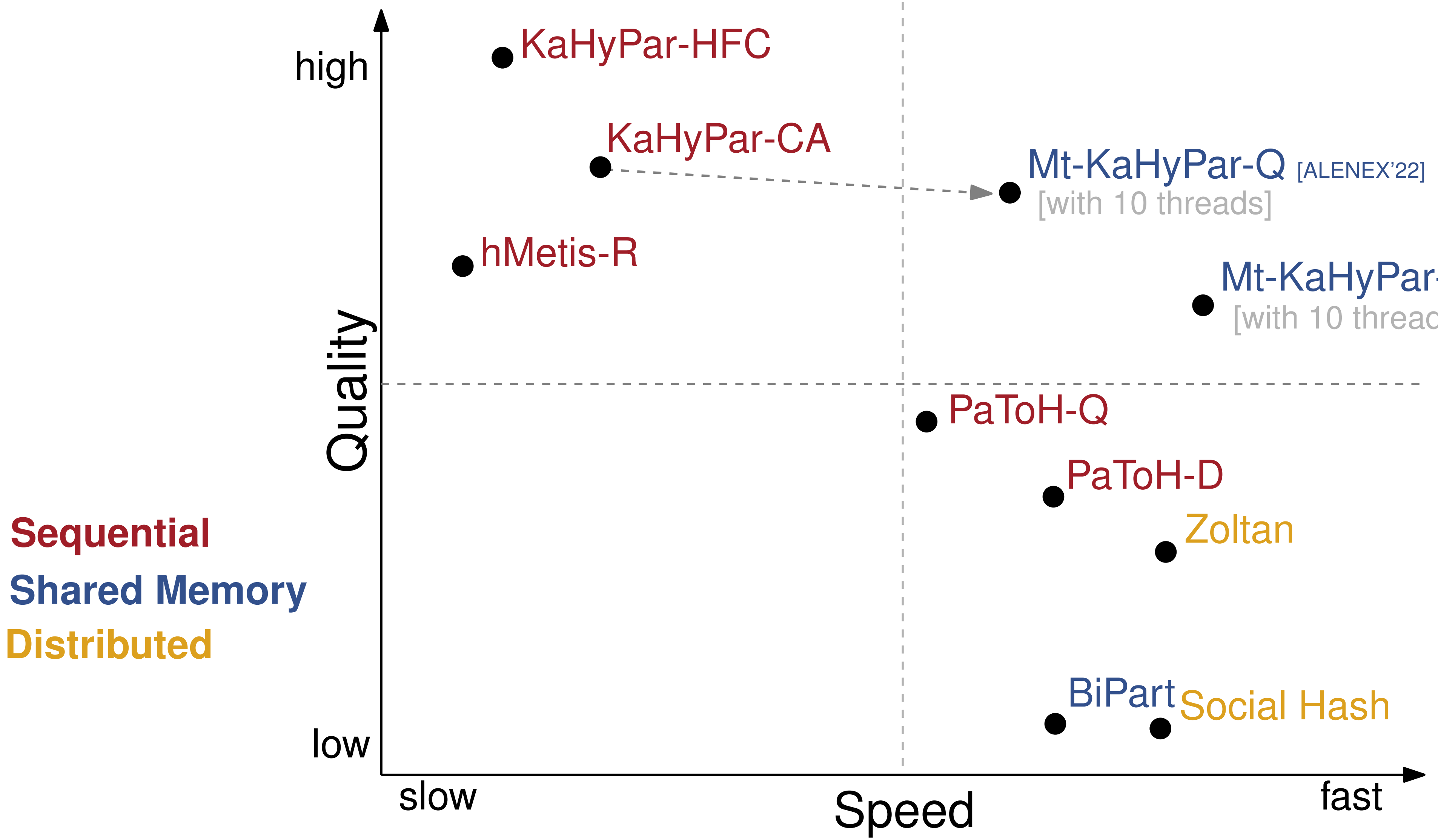
Sequential
Shared Memory
Distributed

Trade-Off Landscape for Hypergraph Partitioning

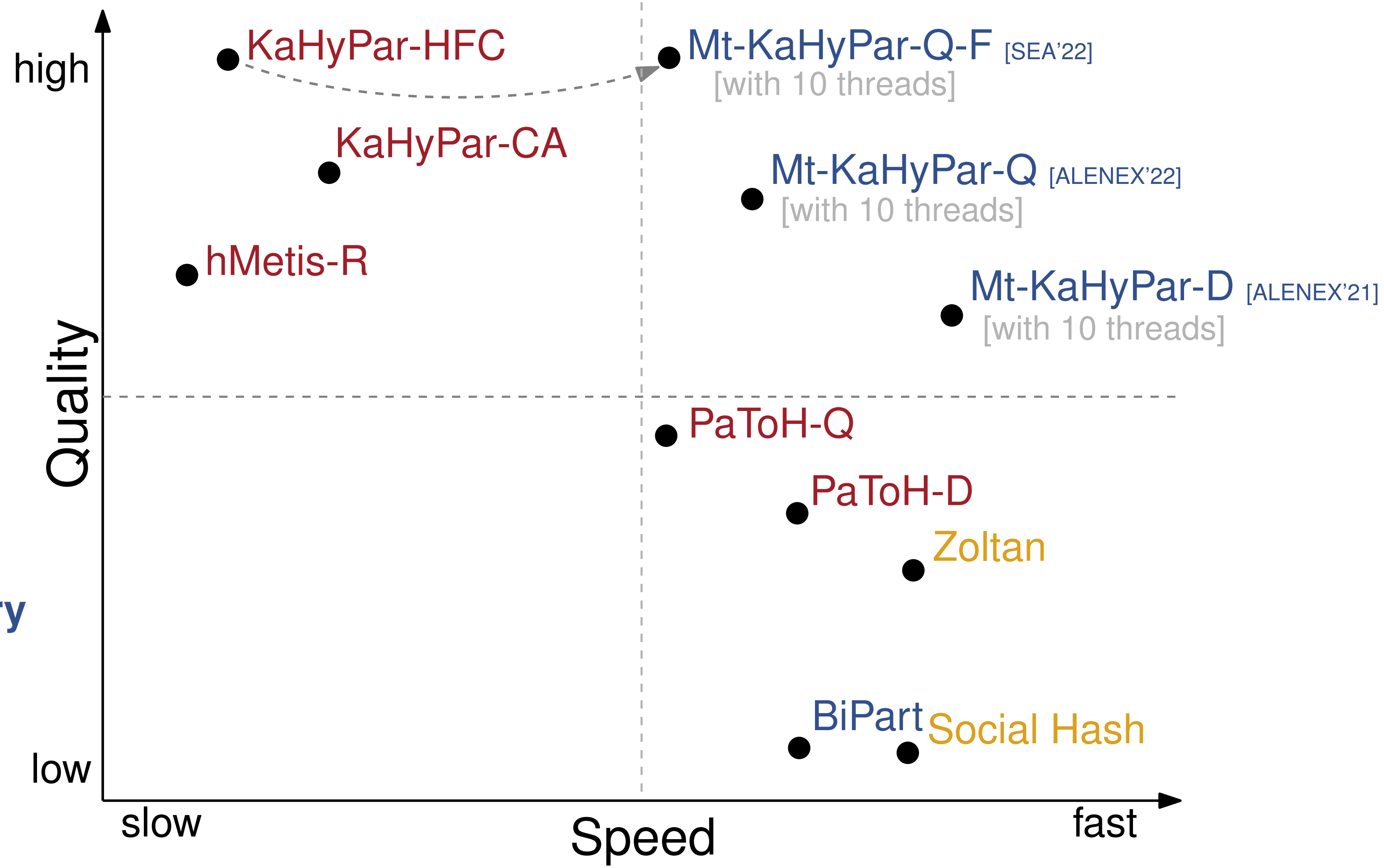


Sequential
Shared Memory
Distributed

Trade-Off Landscape for Hypergraph Partitioning

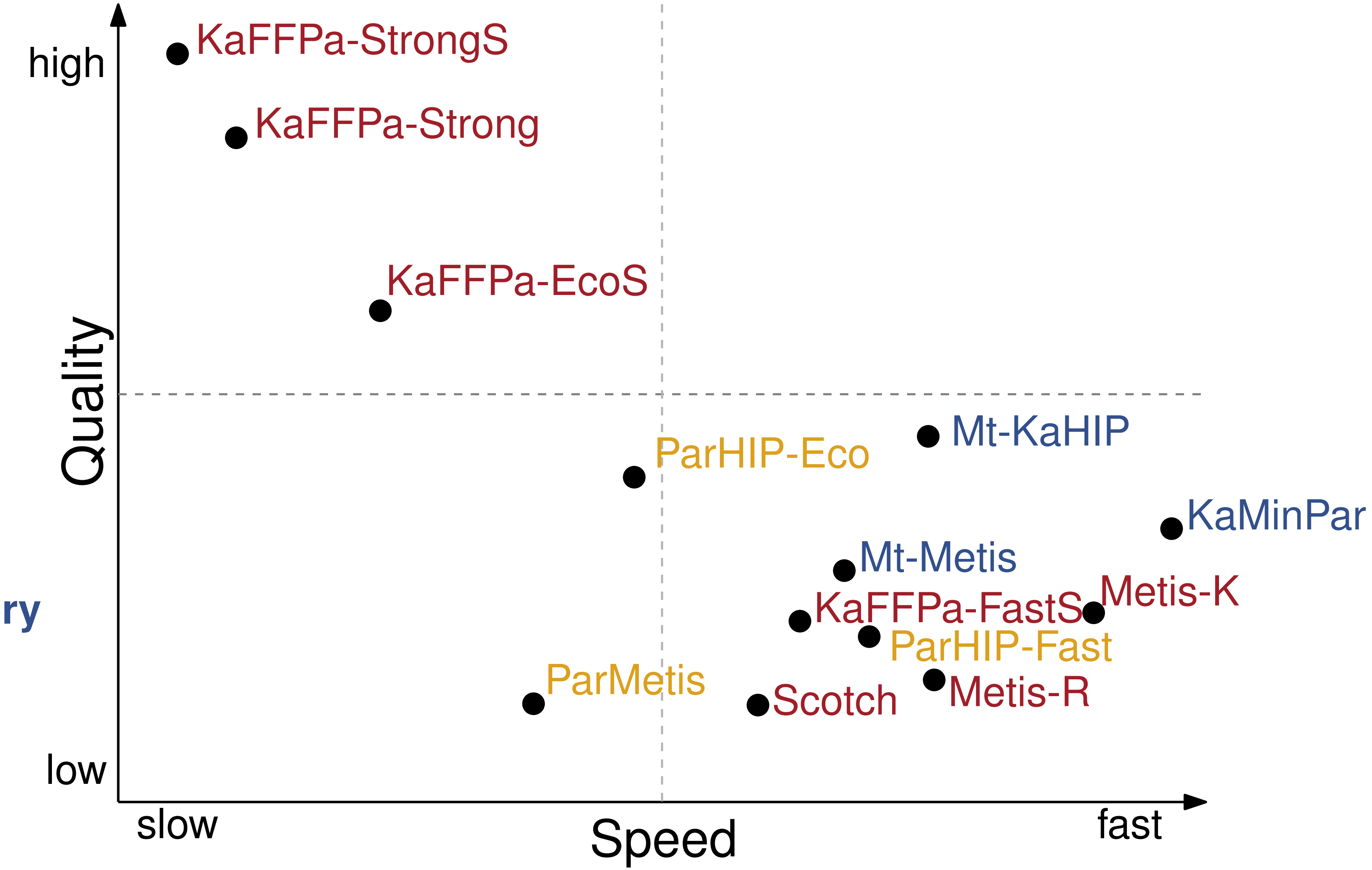


Trade-Off Landscape for Hypergraph Partitioning



Sequential
Shared Memory
Distributed

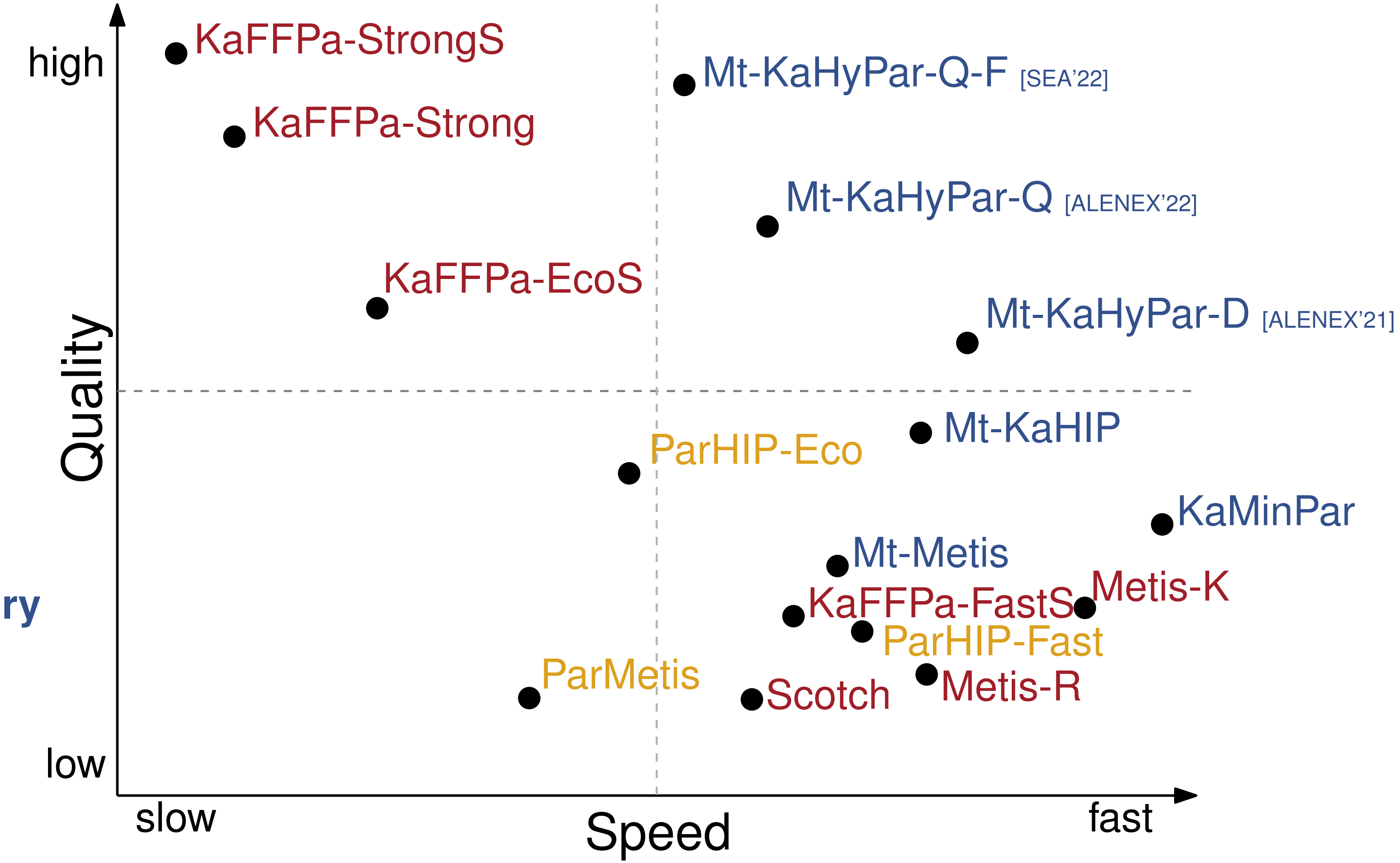
Trade-Off Landscape for Graph Partitioning



Sequential
Shared Memory
Distributed

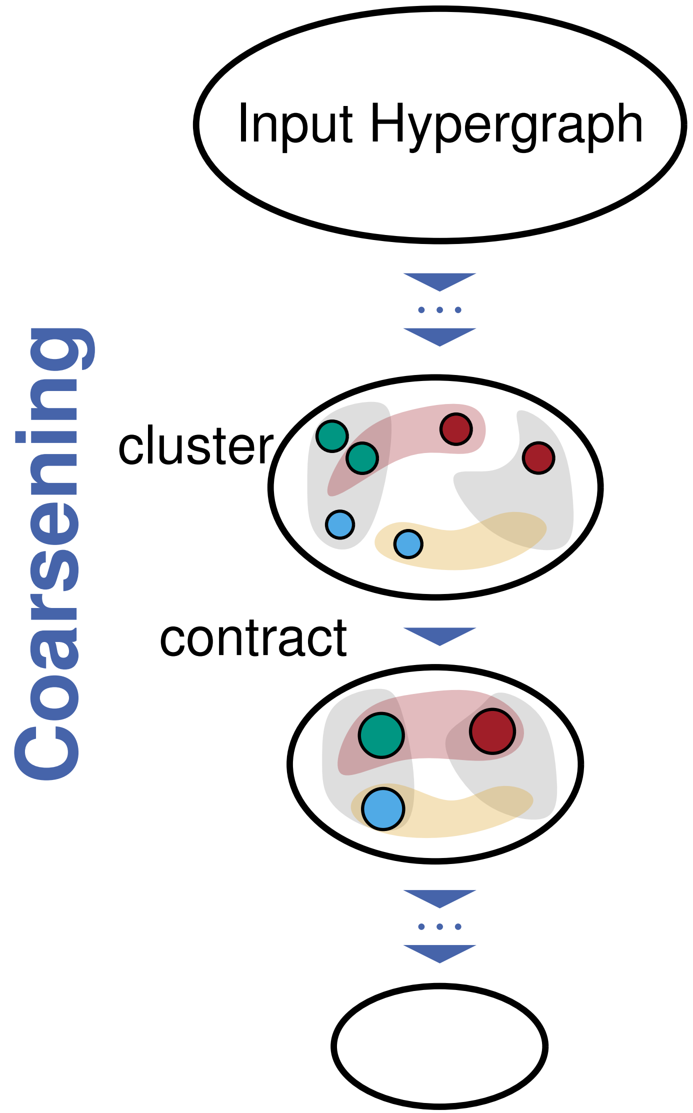
Trade-Off Landscape for Graph Partitioning

Karlsruhe Institute of Technology

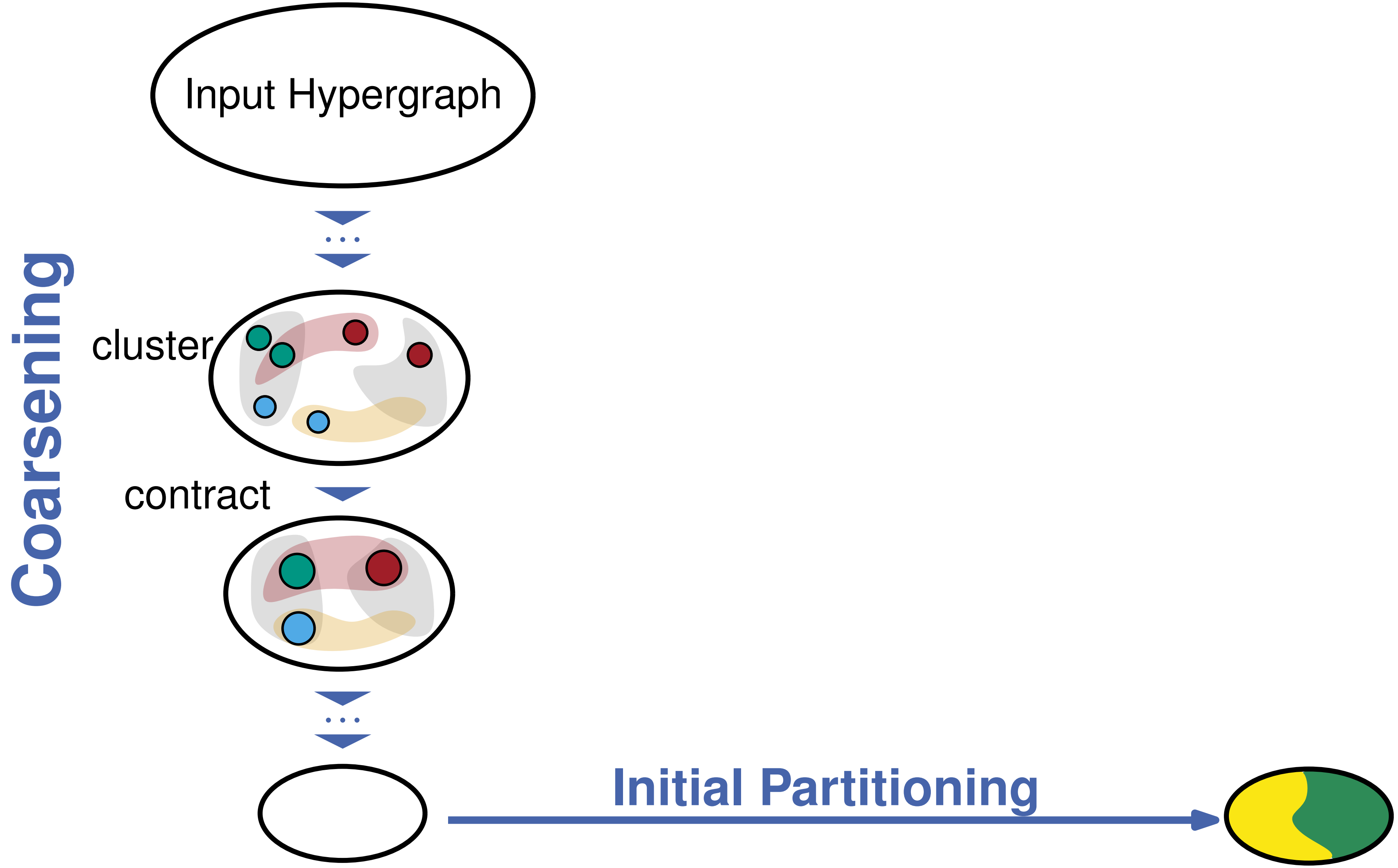


Sequential
Shared Memory
Distributed

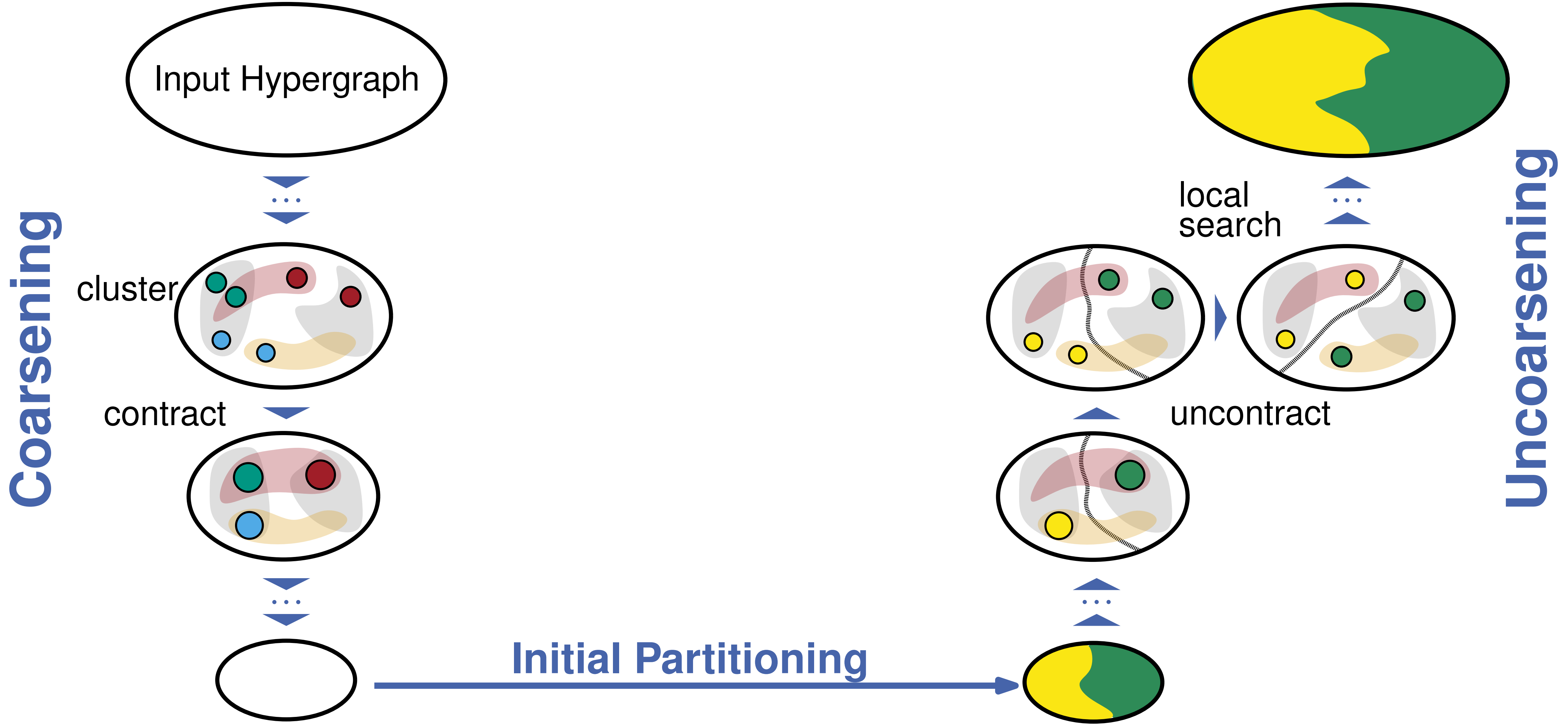
Multilevel Partitioning



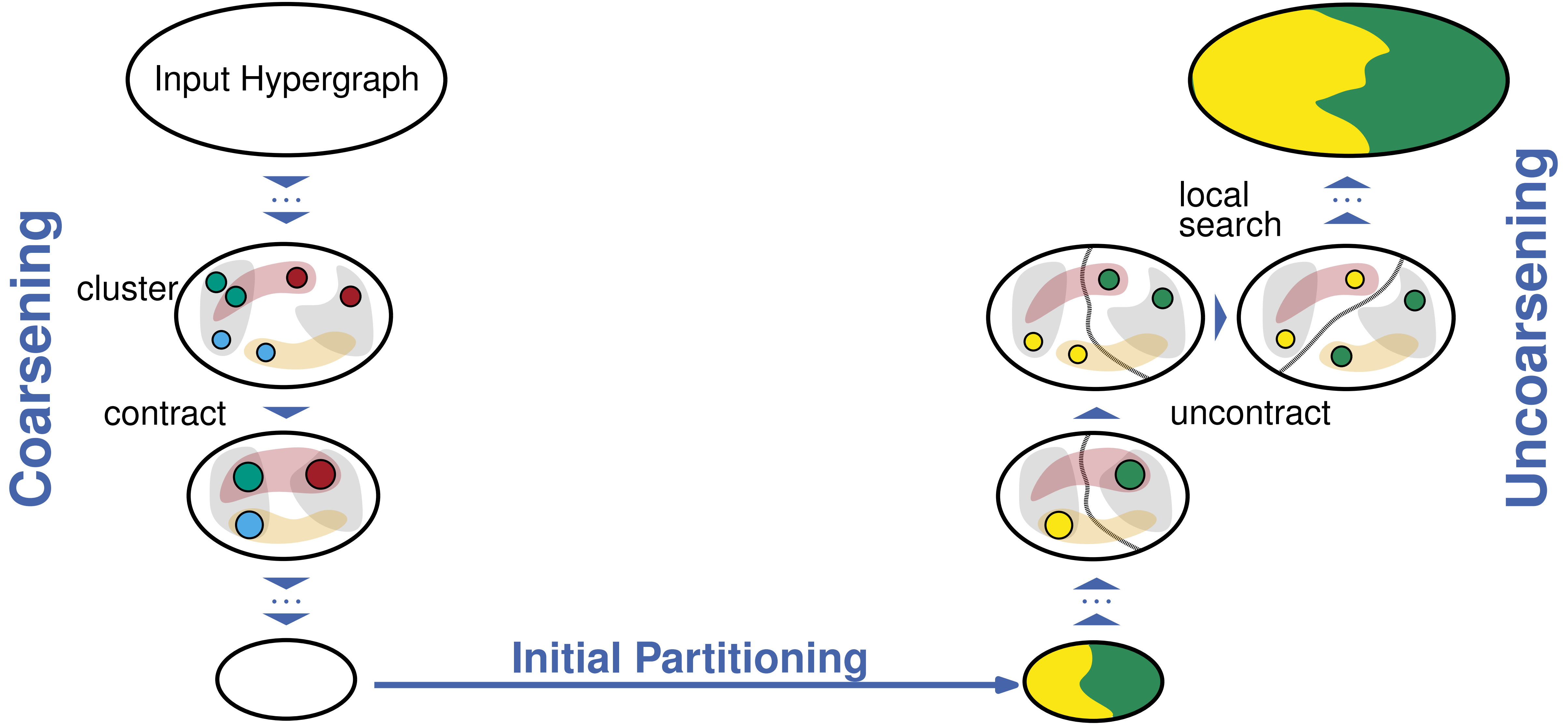
Multilevel Partitioning



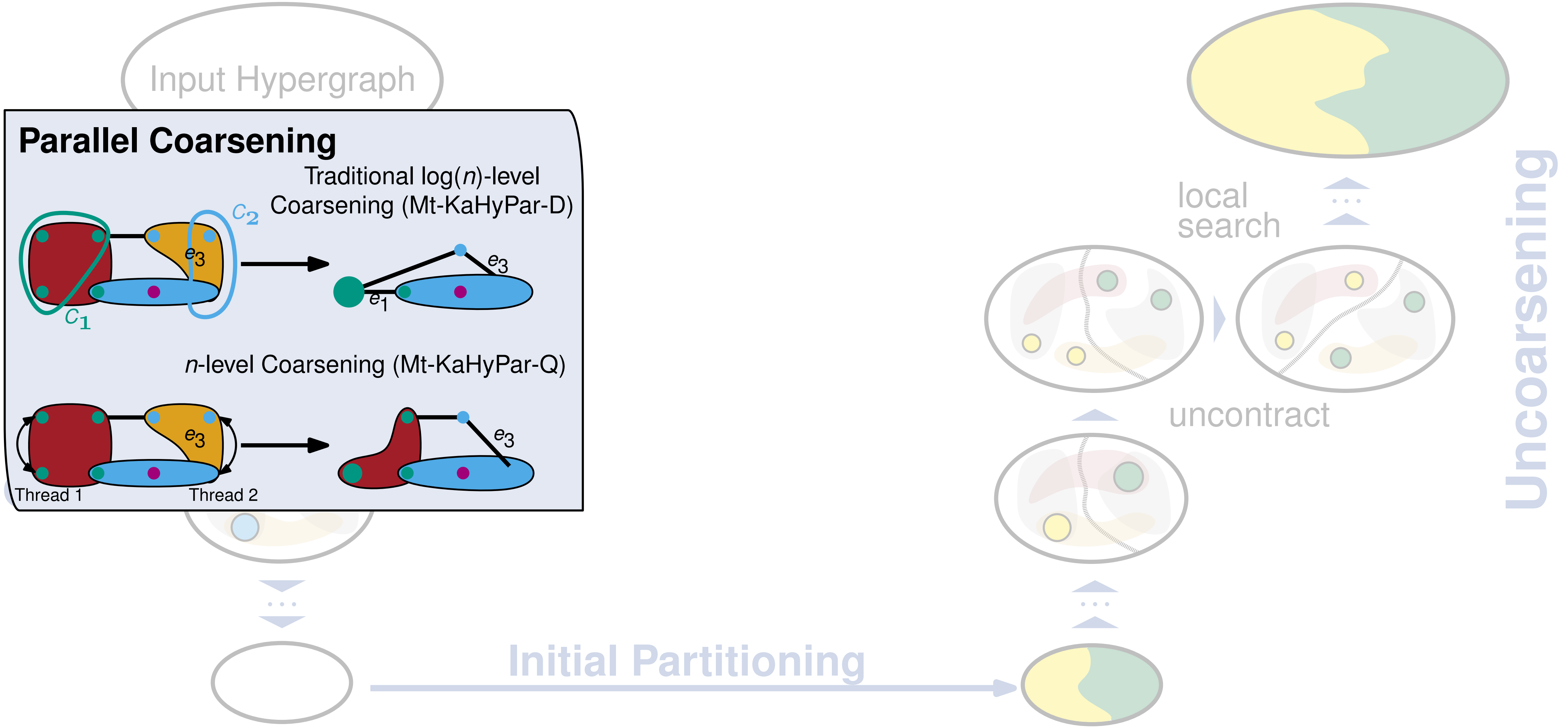
Multilevel Partitioning



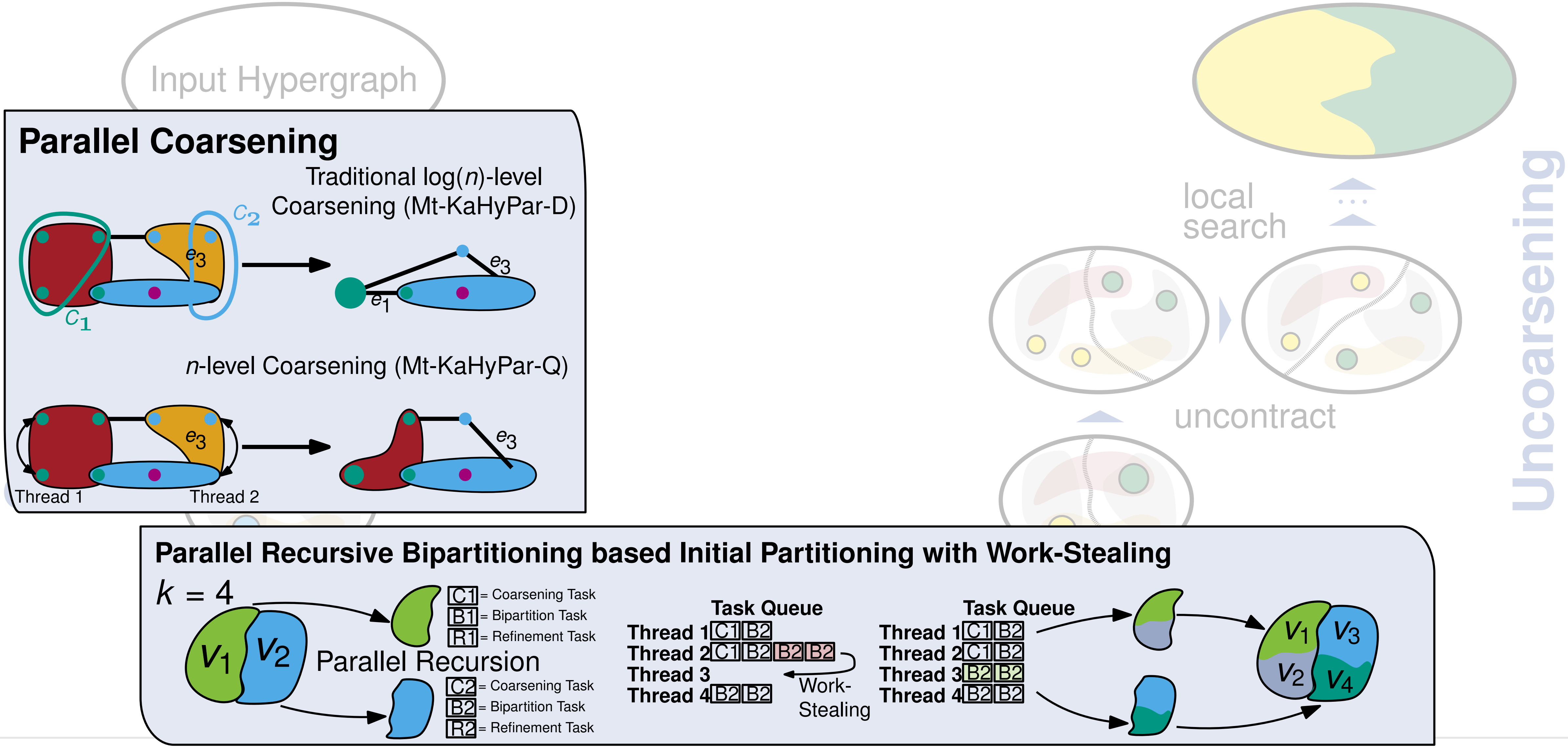
Mt-KaHyPar: Algorithmic Components



Mt-KaHyPar: Algorithmic Components



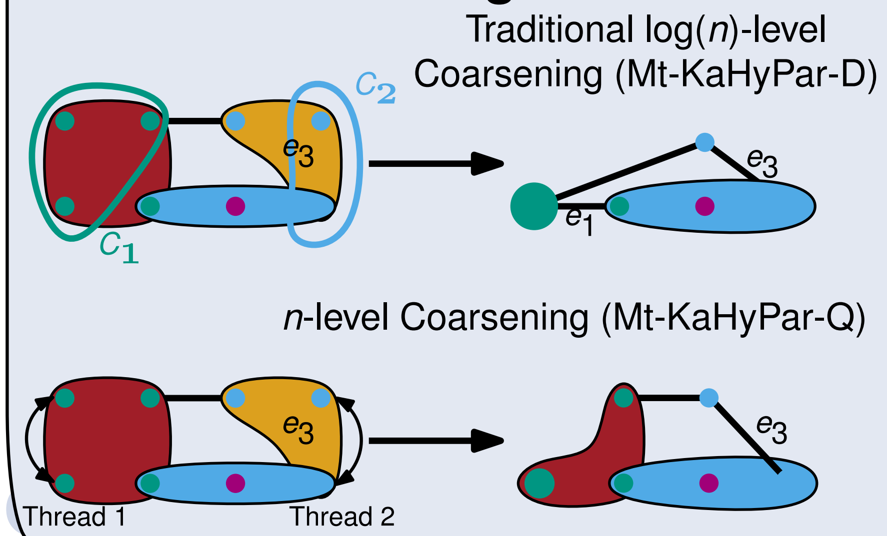
Mt-KaHyPar: Algorithmic Components



Mt-KaHyPar: Algorithmic Components

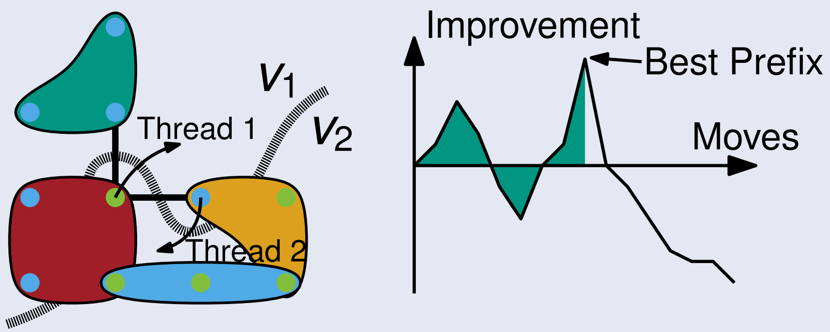
Input Hypergraph

Parallel Coarsening

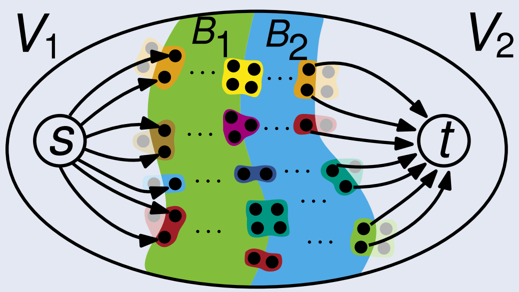


Parallel Direct k-Way FM

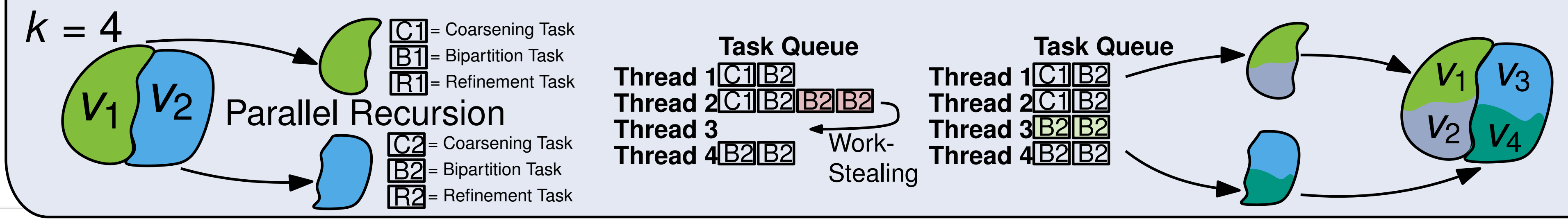
Moves vertices greedily



Parallel Flow-Based Refinement

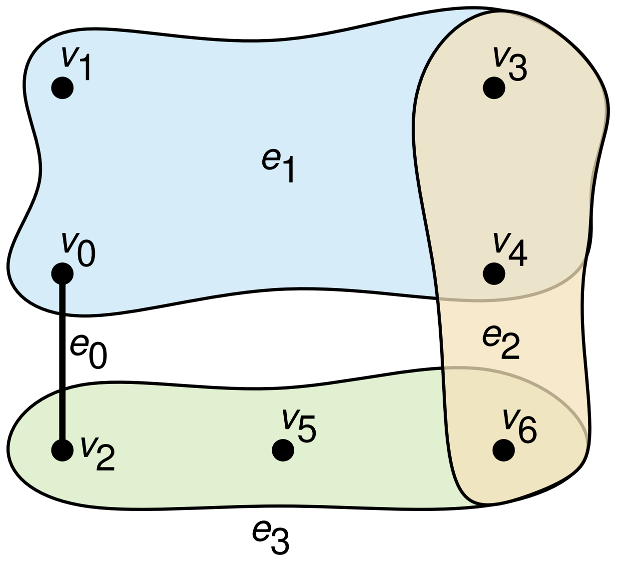


Parallel Recursive Bipartitioning based Initial Partitioning with Work-Stealing



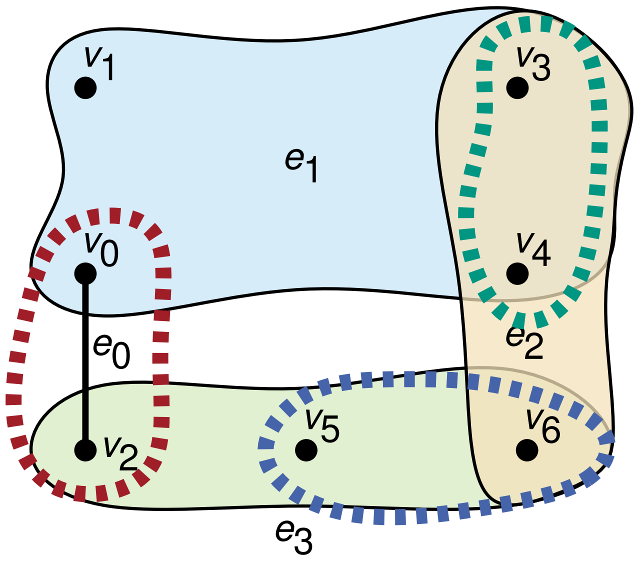
Traditional Multilevel Partitioning

- contracts matching or clustering on each level



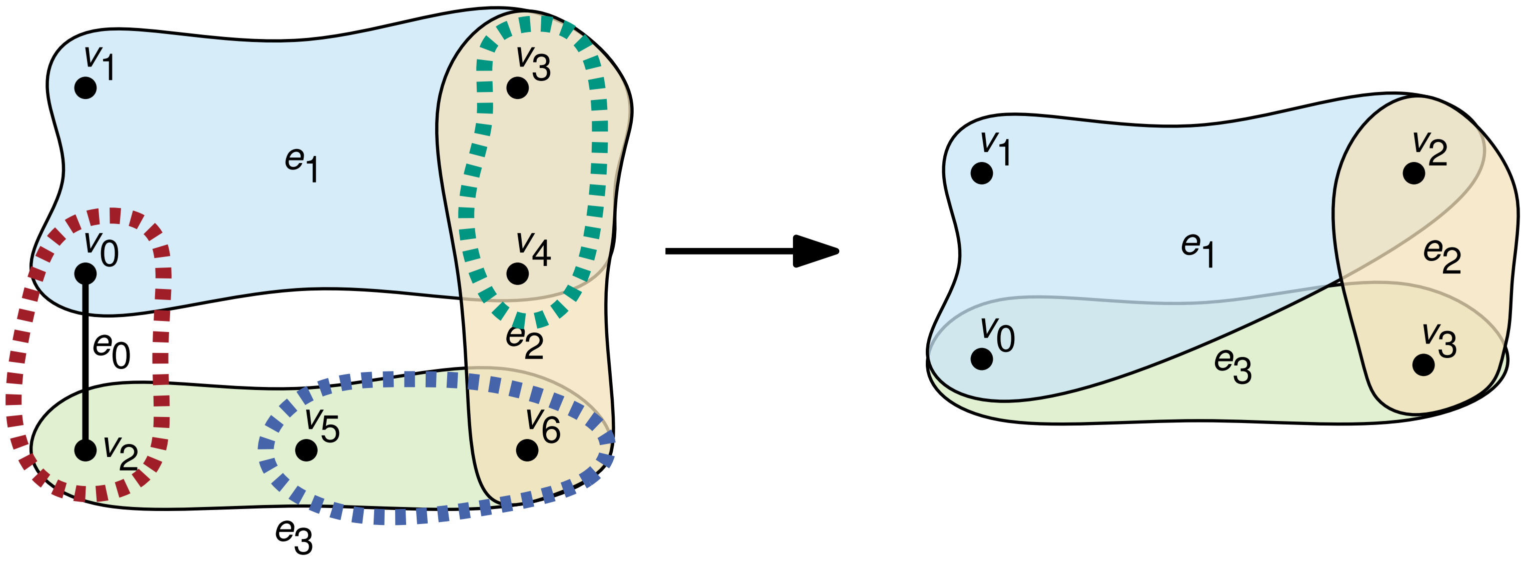
Traditional Multilevel Partitioning

- contracts matching or clustering on each level



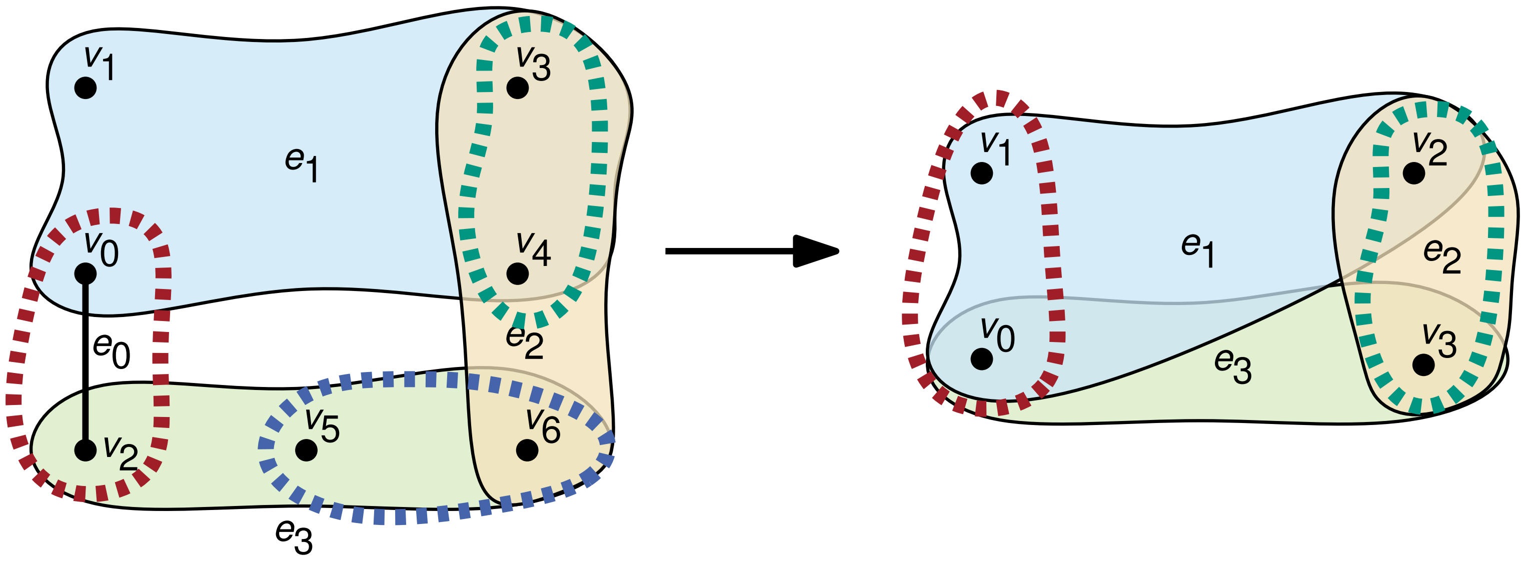
Traditional Multilevel Partitioning

- contracts matching or clustering on each level



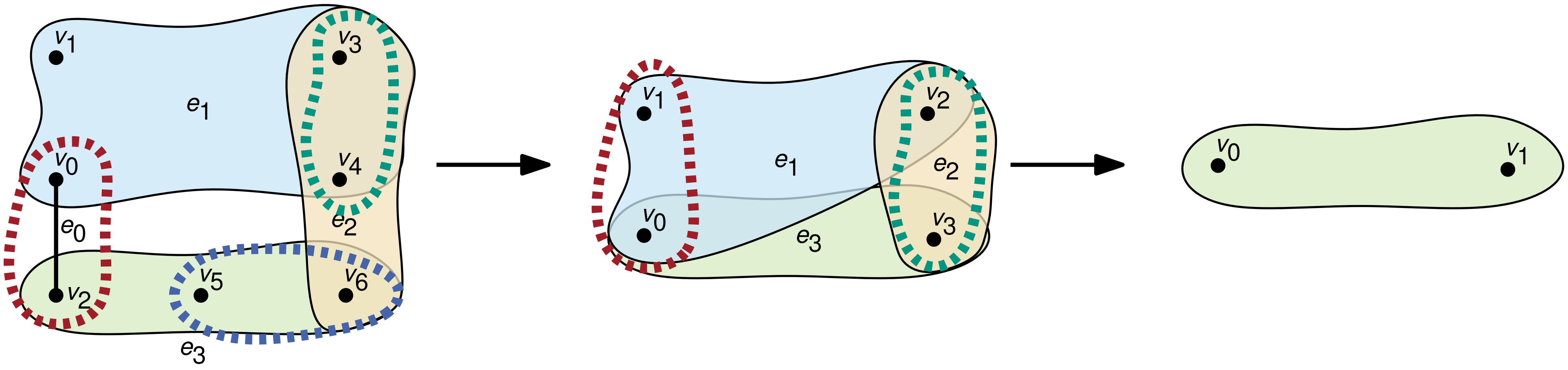
Traditional Multilevel Partitioning

- contracts matching or clustering on each level



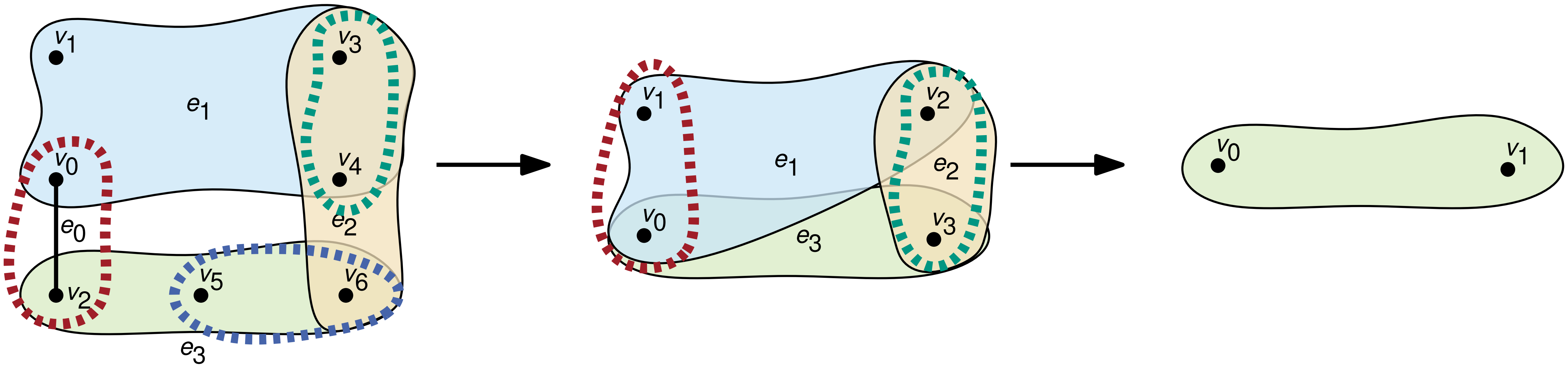
Traditional Multilevel Partitioning

- contracts matching or clustering on each level



Traditional Multilevel Partitioning

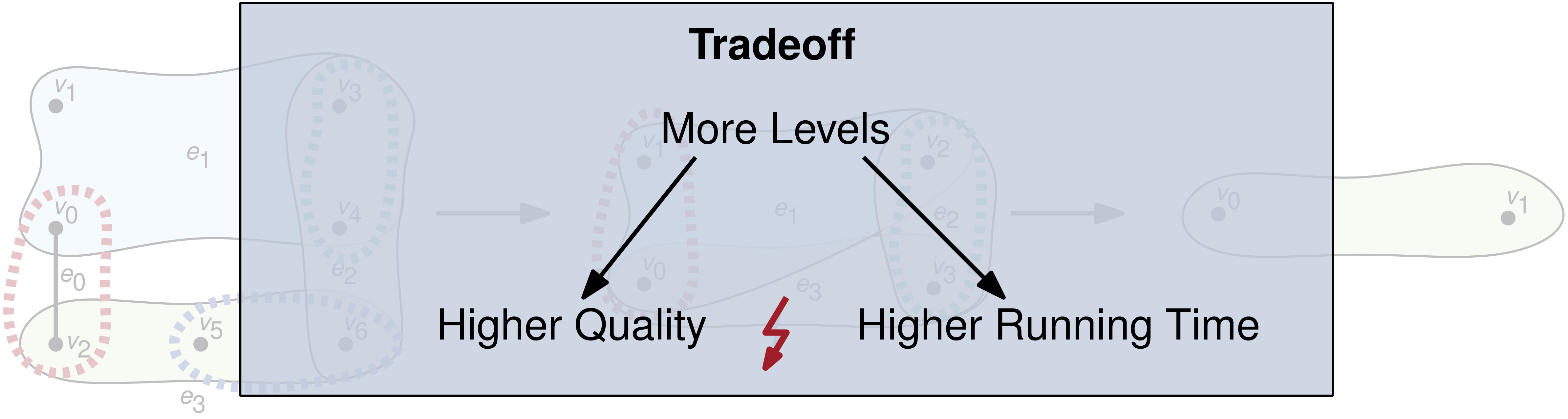
- contracts matching or clustering on each level



$\Rightarrow$ approximately $\mathcal{O}(\log n)$ levels

Traditional Multilevel Partitioning

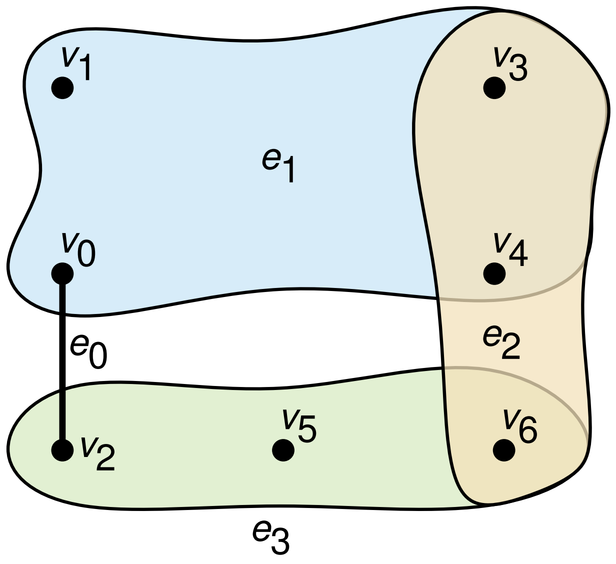
- contracts matching or clustering on each level



$\Rightarrow$ approximately $\mathcal{O}(\log n)$ levels

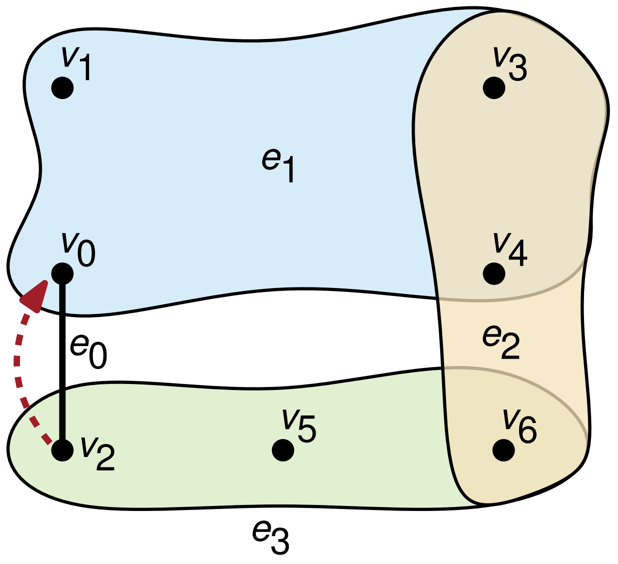
n-level Partitioning

- contract one vertex at a time



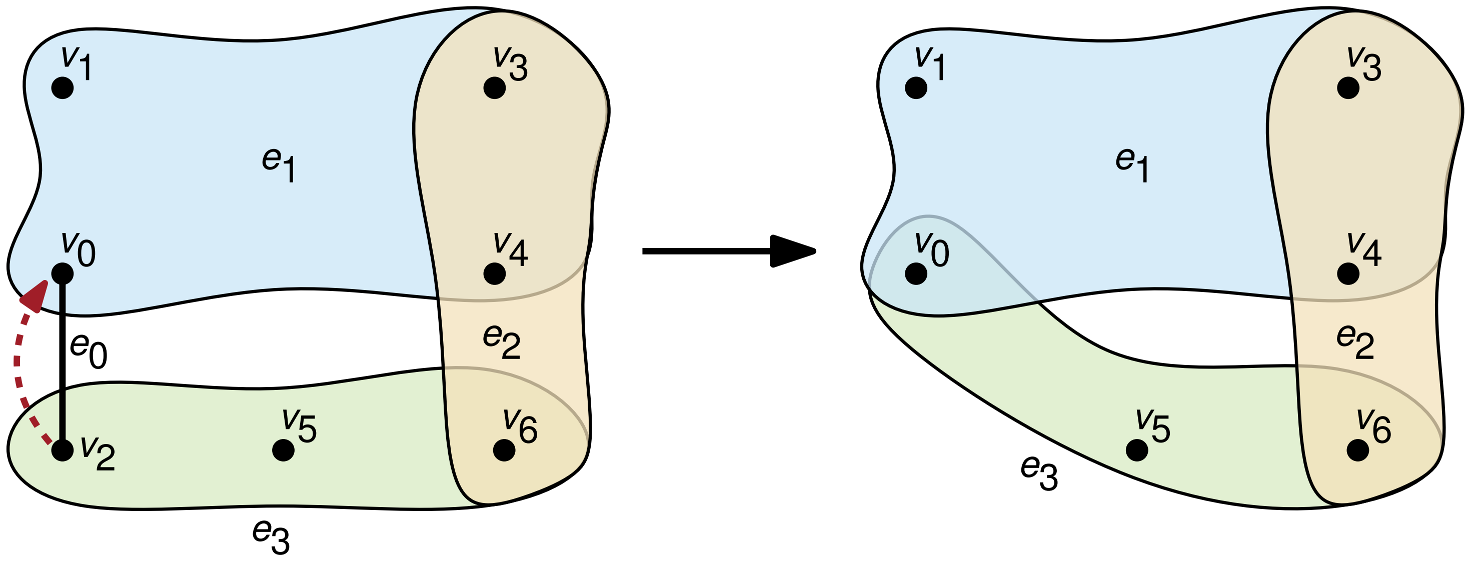
n-level Partitioning

- contract one vertex at a time



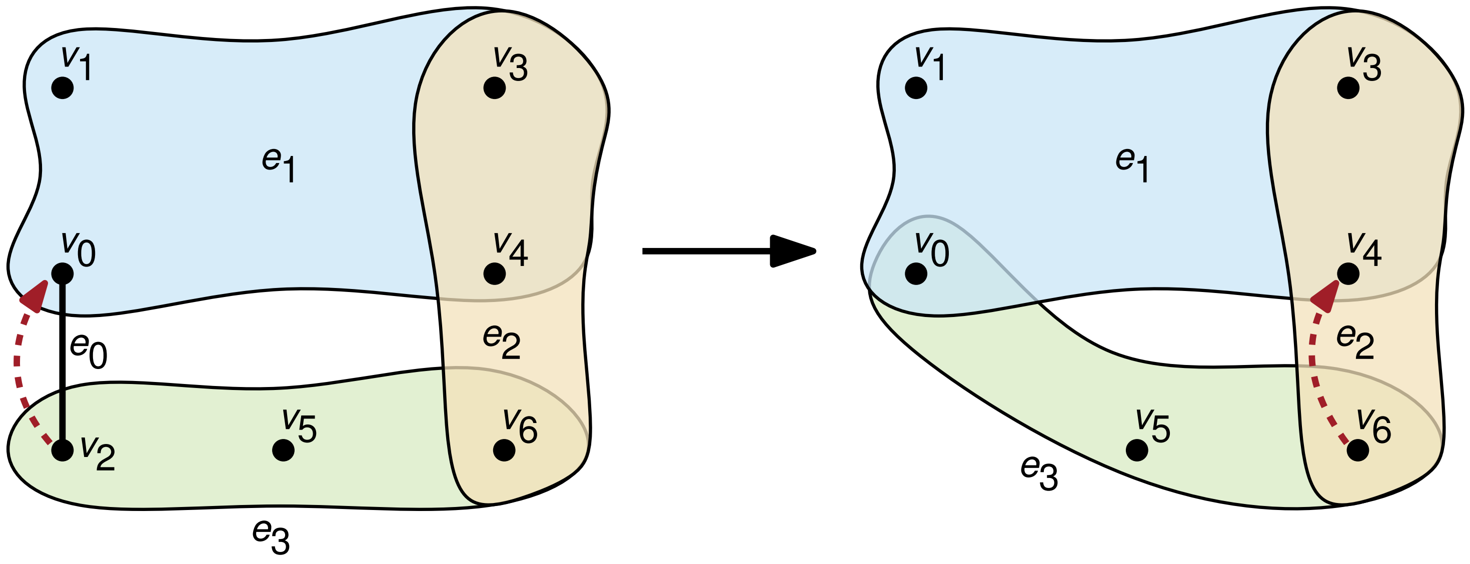
n-level Partitioning

- contract one vertex at a time



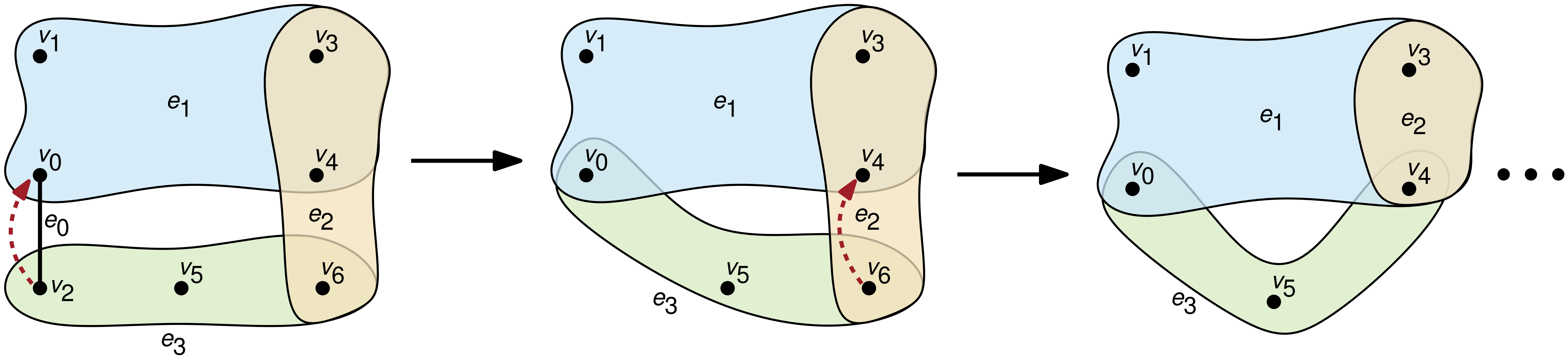
n-level Partitioning

- contract one vertex at a time



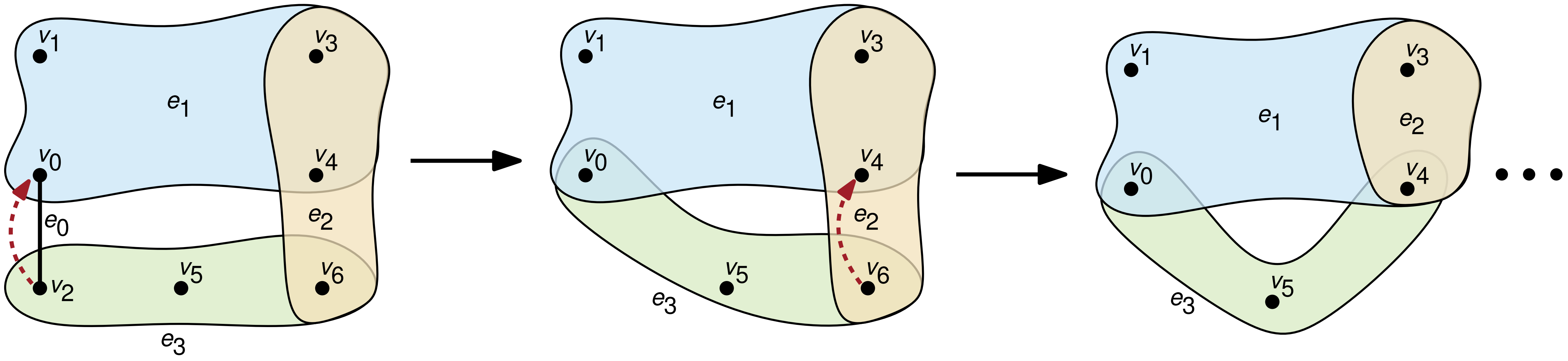
n-level Partitioning

- contract one vertex at a time



n-level Partitioning

- contract one vertex at a time

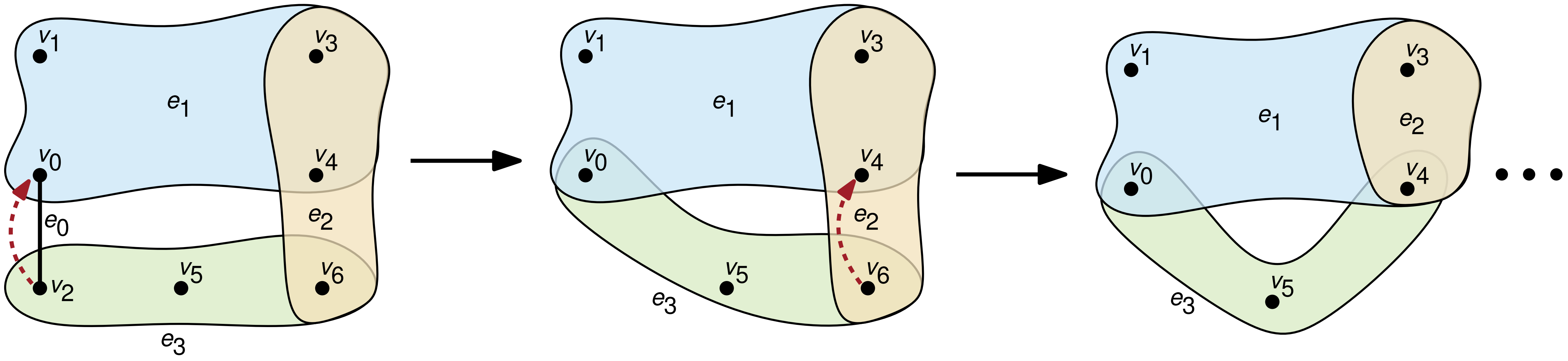


Coarsening: Almost *n* levels

Unoarsening: Almost *n* local search invocations ⇒ **High Quality**! (used in KaHyPar)

n-level Partitioning

- contract one vertex at a time



Coarsening: Almost n levels

Unoarsening: Almost n local search invocations $\Rightarrow$ **High Quality**! (used in KaHyPar)

$\Rightarrow$ **Inherently Sequential!**

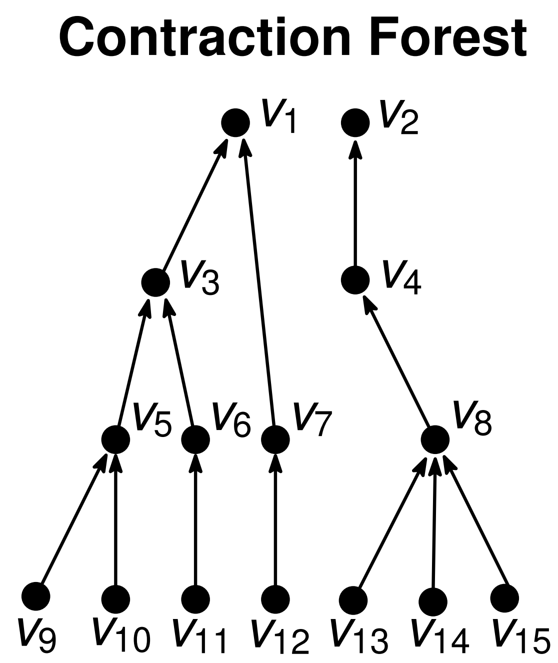
Contraction Forest

Any sequence of contractions form a forest

Contraction Forest

Karlsruhe Institute of Technology

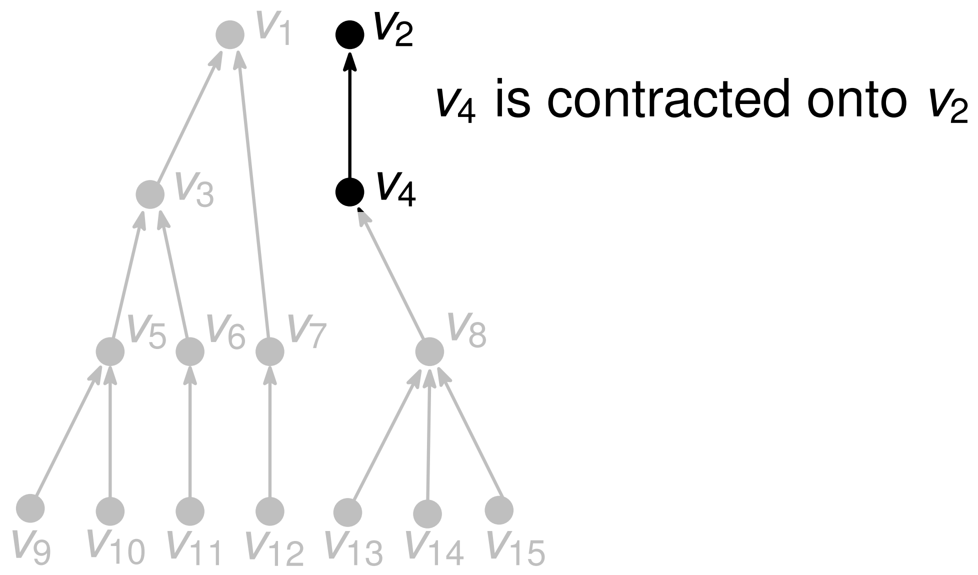
Any sequence of contractions form a forest



Contraction Forest

Any sequence of contractions form a forest

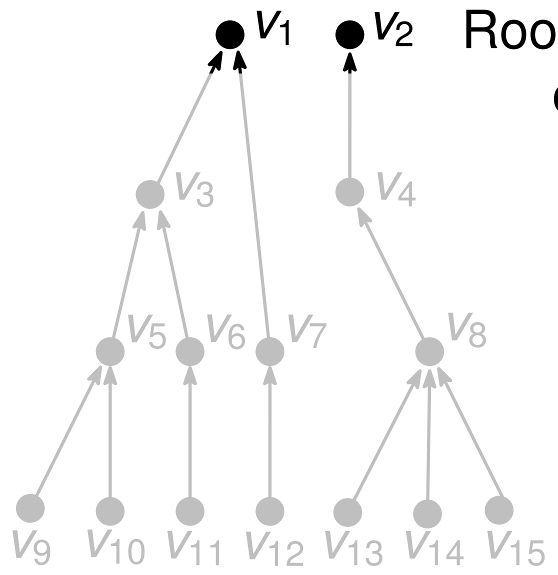
Contraction Forest



Contraction Forest

Any sequence of contractions form a forest

Contraction Forest

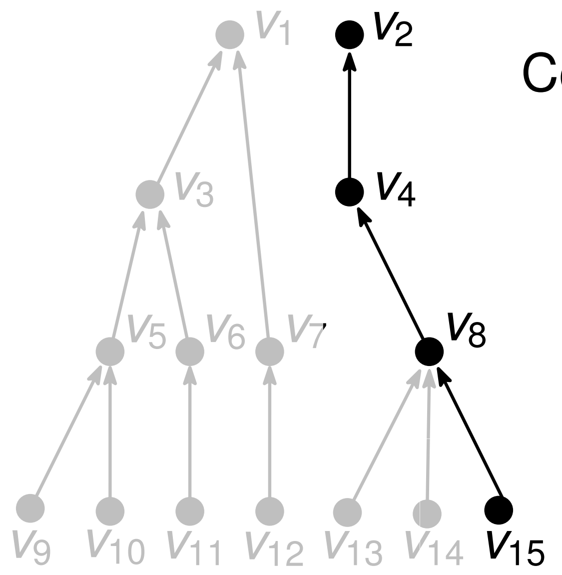


Roots are the vertices of the coarsest hypergraph

Contraction Forest

Any sequence of contractions form a forest

Contraction Forest



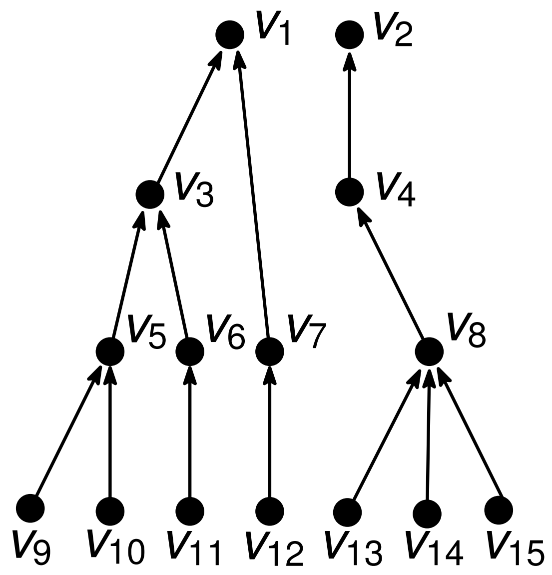
Contraction order:

1. Contract v_{15} onto v_8
2. Contract v_8 onto v_4
3. Contract v_4 onto v_2

Contraction Forest

Any sequence of contractions form a forest

Contraction Forest



Observations

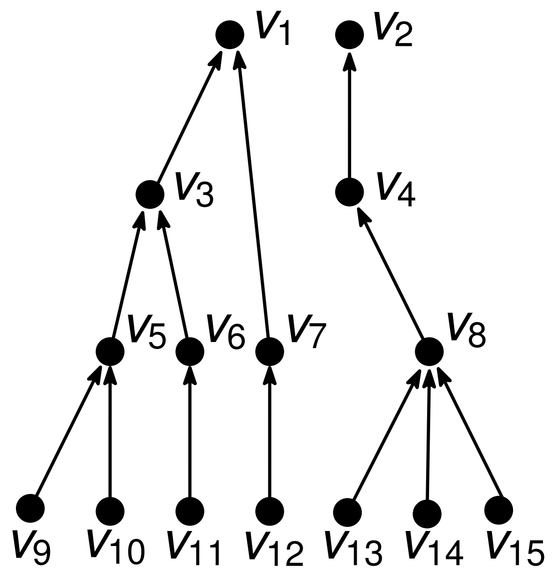
- There is more than one contraction order leading to the same contraction forest

Contraction Forest

Karlsruhe Institute of Technology

Any sequence of contractions form a forest

Contraction Forest



Observations

- There is more than one contraction order leading to the same contraction forest

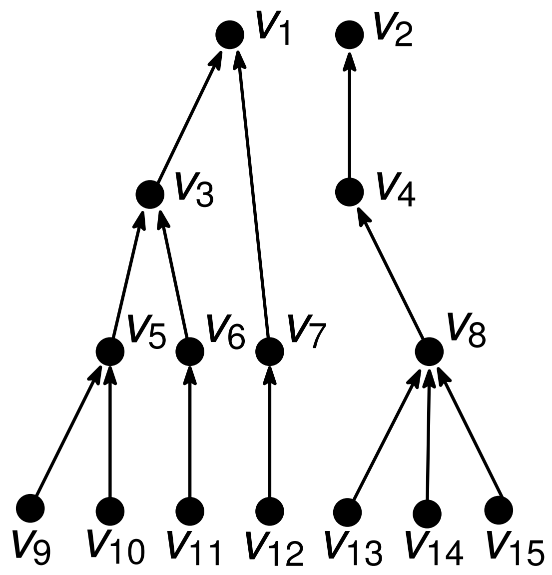
Rules

- Contractions in different subtrees are independent
- Contract v when its children are contracted onto v

Contraction Forest

Any sequence of contractions form a forest

Contraction Forest



Observations

- There is more than one contraction order leading to the same contraction forest

Rules

- Contractions in different subtrees are independent
- Contract v when its children are contracted onto v

Parallelization Idea

- Contract contraction forest bottom-up in parallel

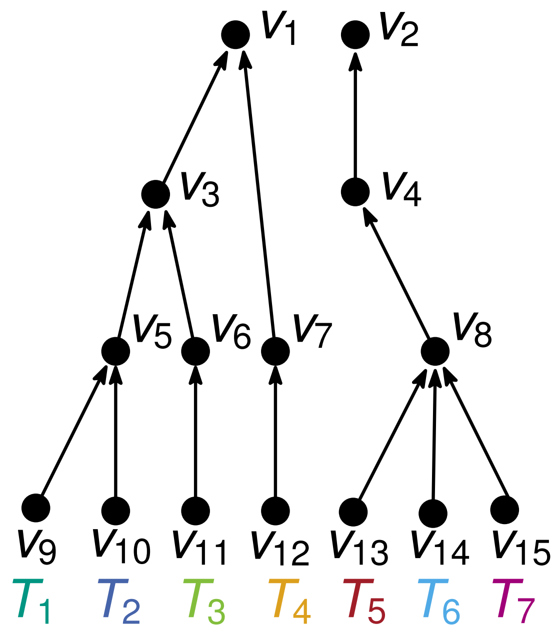
Contraction Forest

Karlsruhe Institute of Technology

Any sequence of contractions form a forest

T_i = Thead i

Contraction Forest



Observations

- There is more than one contraction order leading to the same contraction forest

Rules

- Contractions in different subtrees are independent
- Contract v when its children are contracted onto v

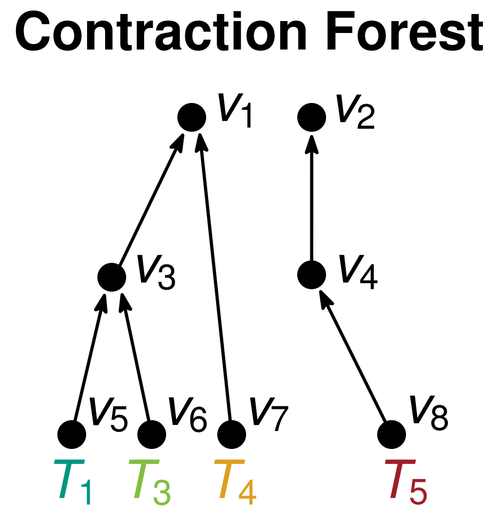
Parallelization Idea

- Contract contraction forest bottom-up in parallel

Contraction Forest

Any sequence of contractions form a forest

T_i = Thead i



Observations

- There is more than one contraction order leading to the same contraction forest

Rules

- Contractions in different subtrees are independent
- Contract v when its children are contracted onto v

Parallelization Idea

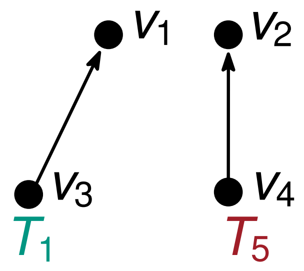
- Contract contraction forest bottom-up in parallel

Contraction Forest

Any sequence of contractions form a forest

T_i = Thead i

Contraction Forest



Observations

- There is more than one contraction order leading to the same contraction forest

Rules

- Contractions in different subtrees are independent
- Contract v when its children are contracted onto v

Parallelization Idea

- Contract contraction forest bottom-up in parallel

Contraction Forest

Any sequence of contractions form a forest

Contraction Forest

T_i = Thead i

Observations

- There is more than one contraction order leading to the same contraction forest

Rules

- Contractions in different subtrees are independent
- Contract v when its children are contracted onto v

Parallelization Idea

- Contract contraction forest bottom-up in parallel

Contraction Forest

Any sequence of contractions form a forest

Contraction Forest

T_i = Thead i

Observations

- There is more than one contraction order leading to the same contraction forest

Rules

- Contractions in different subtrees are independent
- Contract v when its children are contracted onto v

Parallelization Idea

- Contract contraction forest bottom-up in parallel

Problem: Contraction forest is not known in advance

Contraction Forest Construction

Idea: Construct contraction forest *on-the-fly*

Algorithm 1: Parallel n-level Coarsening

for each $u \in V$ **in parallel**
    $v \leftarrow$ find contraction partner for u
    if *add* (v, u) *to contraction forest* **then**
        contract v onto u

Contraction Forest Construction

Karlsruhe Institute of Technology

Idea: Construct contraction forest *on-the-fly*

Algorithm 1: Parallel n-level Coarsening

for each $u \in V$ **in parallel**
    $v \leftarrow$ find contraction partner for u
    if *add* (v, u) *to contraction forest* **then**
        contract v onto u

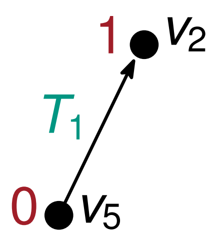
Contraction Forest Construction

Idea: Construct contraction forest *on-the-fly*

Algorithm 1: Parallel n-level Coarsening

for each $u \in V$ **in parallel**
  $v \leftarrow$ find contraction partner for u
  if *add* (v, u) *to contraction forest* **then**
    contract v onto u

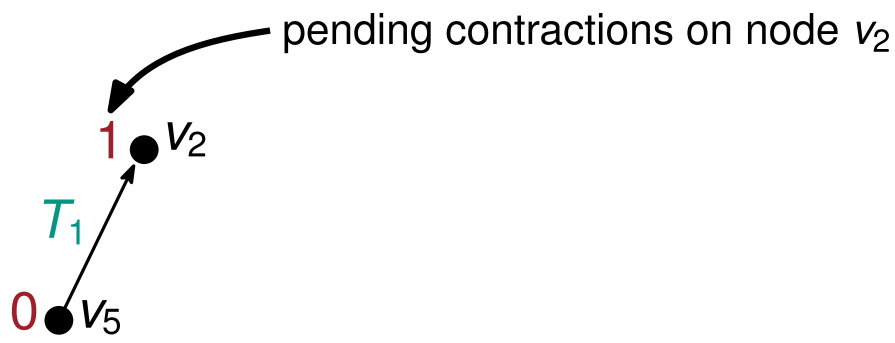
T_i = Thead i



Contraction Forest Construction

Idea: Construct contraction forest *on-the-fly*

T_i = Thead i



Algorithm 1: Parallel *n*-level Coarsening

for each $u \in V$ **in parallel**
  $v \leftarrow$ find contraction partner for u
  if *add* (v, u) *to contraction forest* **then**
    contract v onto u

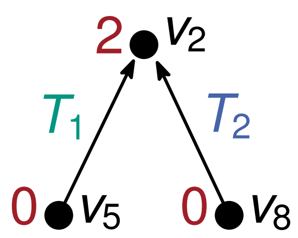
Contraction Forest Construction

Idea: Construct contraction forest *on-the-fly*

Algorithm 1: Parallel n-level Coarsening

for each $u \in V$ **in parallel**
  $v \leftarrow$ find contraction partner for u
  if *add* (v, u) *to contraction forest* **then**
    contract v onto u

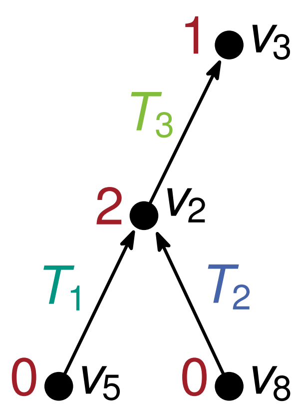
T_i = Thead i



Contraction Forest Construction

Idea: Construct contraction forest *on-the-fly*

T_i = Thead i



Algorithm 1: Parallel n-level Coarsening

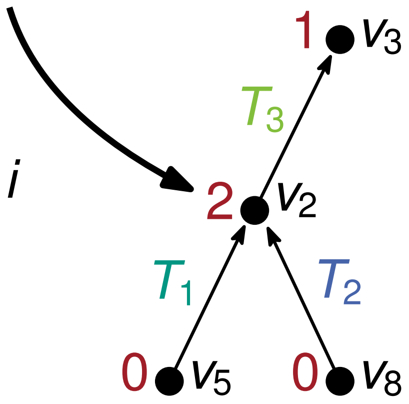
for each $u \in V$ **in parallel**
    $v \leftarrow$ find contraction partner for u
    if *add* (v, u) *to contraction forest* **then**
        contract v onto u

Contraction Forest Construction

Idea: Construct contraction forest *on-the-fly*

(v_2, v_3) is not eligible for contraction
$\Rightarrow$ do something else

T_i = Thead i



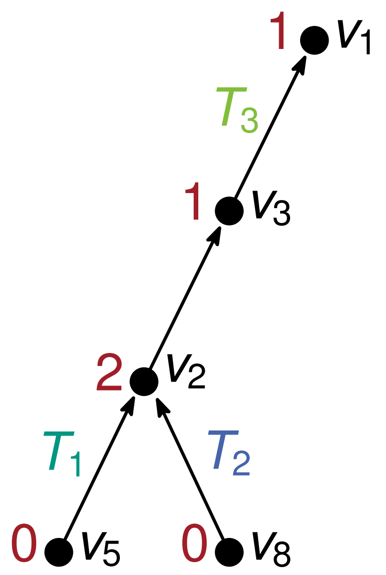
Algorithm 1: Parallel n-level Coarsening

for each $u \in V$ **in parallel**
  $v \leftarrow$ find contraction partner for u
  if *add* (v, u) *to contraction forest* **then**
    contract v onto u

Contraction Forest Construction

Idea: Construct contraction forest *on-the-fly*

T_i = Thead i



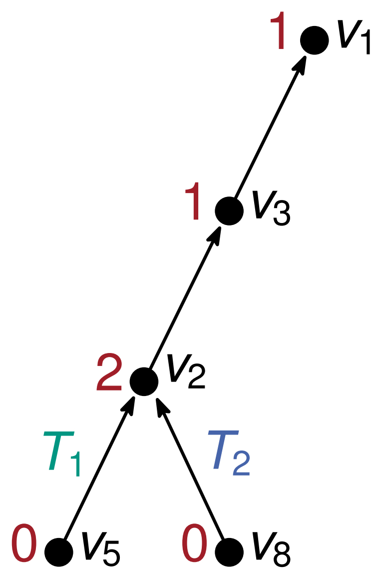
Algorithm 1: Parallel n-level Coarsening

for each $u \in V$ **in parallel**
  $v \leftarrow$ find contraction partner for u
  if *add* (v, u) *to contraction forest* **then**
    contract v onto u

Contraction Forest Construction

Idea: Construct contraction forest *on-the-fly*

T_i = Thead *i*



Algorithm 1: Parallel *n*-level Coarsening

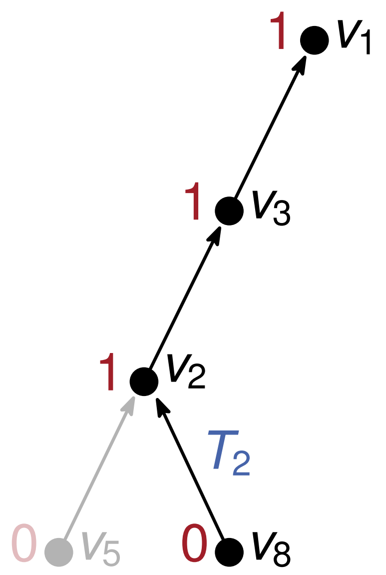
for each $u \in V$ **in parallel**
  $v \leftarrow$ find contraction partner for u
  if *add* (v, u) *to contraction forest* **then**
    contract v onto u

Contraction Forest Construction

Karlsruhe Institute of Technology

Idea: Construct contraction forest *on-the-fly*

T_i = Thead i



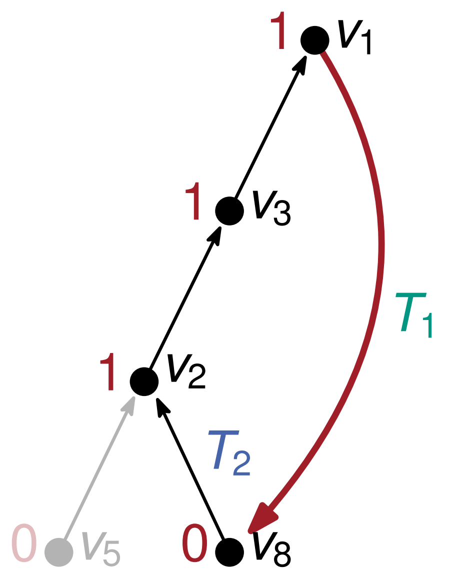
Algorithm 1: Parallel n-level Coarsening

for each $u \in V$ **in parallel**
  $v \leftarrow$ find contraction partner for u
  if *add* (v, u) *to contraction forest* **then**
    contract v onto u

Contraction Forest Construction

Idea: Construct contraction forest *on-the-fly*

T_i = Thead i



Cyclic Dependency
⇒ Discard Contraction

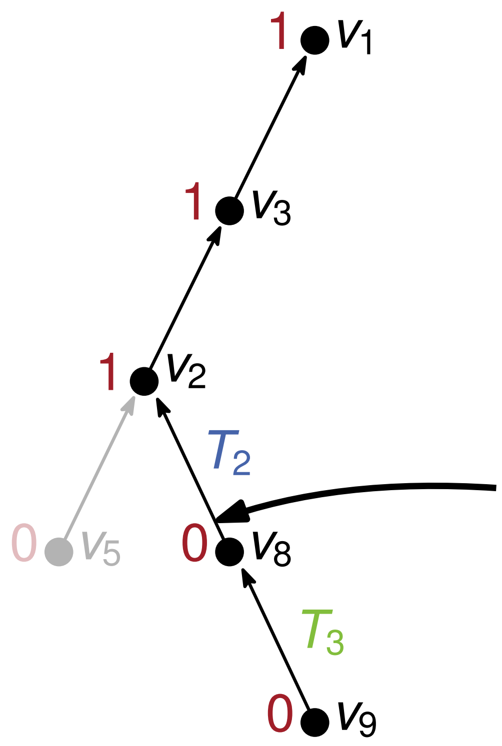
Algorithm 1: Parallel *n*-level Coarsening

for each $u \in V$ **in parallel**
  $v \leftarrow$ find contraction partner for u
  if *add* (v, u) *to contraction forest* **then**
    contract v onto u

Contraction Forest Construction

Idea: Construct contraction forest *on-the-fly*

T_i = Thead i



Pending counter of v_8 is **zero**
$\Rightarrow$ we assume contraction of v_8 has already started
$\Rightarrow$ find suitable ancestor of v_8

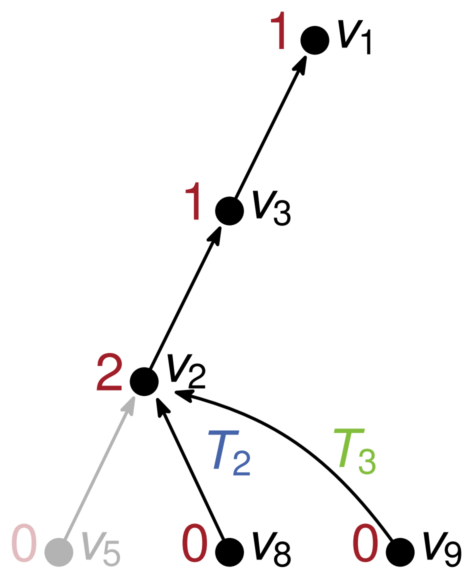
Algorithm 1: Parallel n-level Coarsening

for each $u \in V$ **in parallel**
  $v \leftarrow$ find contraction partner for u
  if *add* (v, u) *to contraction forest* **then**
    contract v onto u

Contraction Forest Construction

Idea: Construct contraction forest *on-the-fly*

T_i = Thead i

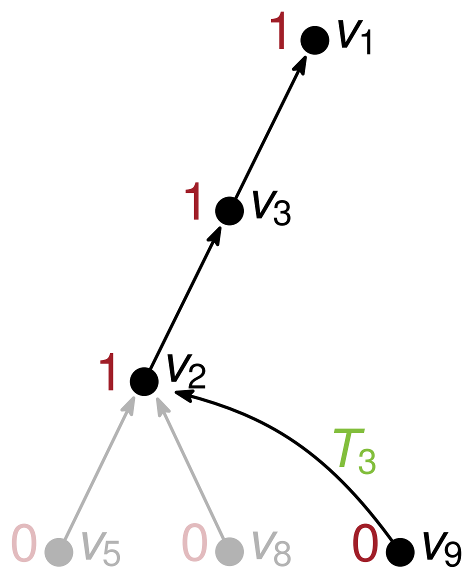


Algorithm 1: Parallel n-level Coarsening

for each $u \in V$ **in parallel**
  $v \leftarrow$ find contraction partner for u
  if *add* (v, u) *to contraction forest* **then**
    contract v onto u

Contraction Forest Construction

Idea: Construct contraction forest *on-the-fly*



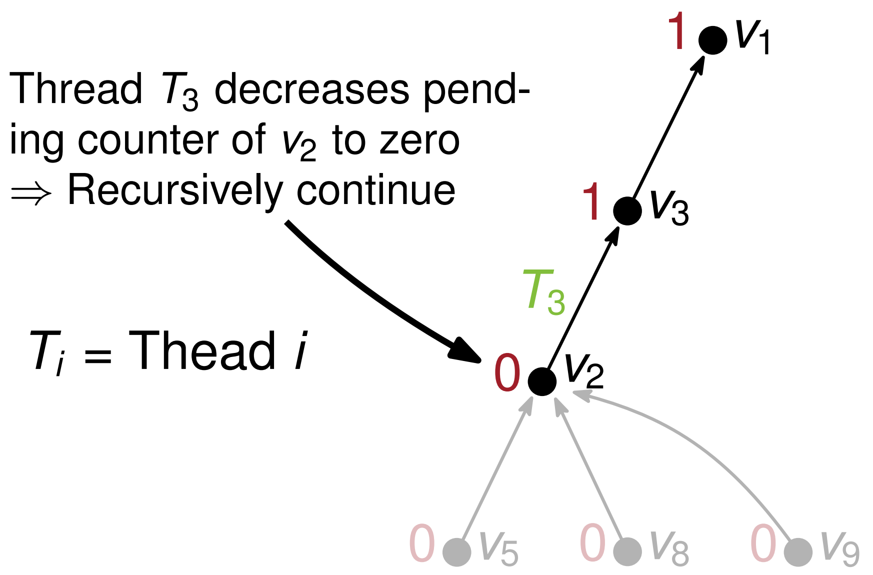
T_i = Thead i

Algorithm 1: Parallel n-level Coarsening

for each $u \in V$ **in parallel**
  $v \leftarrow$ find contraction partner for u
  if *add* (v, u) *to contraction forest* **then**
    contract v onto u

Contraction Forest Construction

Idea: Construct contraction forest *on-the-fly*



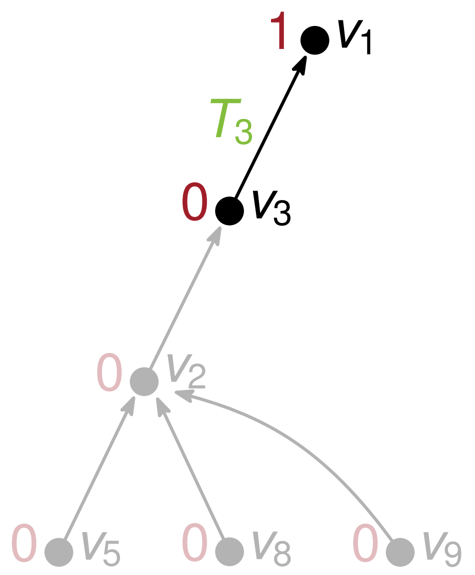
Algorithm 1: Parallel n-level Coarsening

for each $u \in V$ **in parallel**
  $v \leftarrow$ find contraction partner for u
  if *add* (v, u) *to contraction forest* **then**
    contract v onto u

Contraction Forest Construction

Idea: Construct contraction forest *on-the-fly*

T_i = Thead i



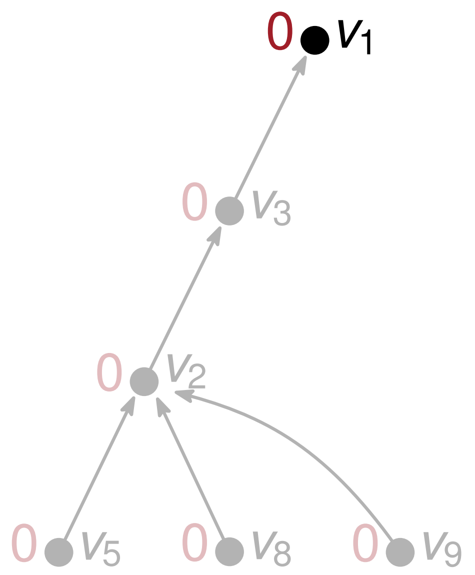
Algorithm 1: Parallel n-level Coarsening

for each $u \in V$ **in parallel**
  $v \leftarrow$ find contraction partner for u
  if *add* (v, u) *to contraction forest* **then**
    contract v onto u

Contraction Forest Construction

Idea: Construct contraction forest *on-the-fly*

T_i = Thead i



Algorithm 1: Parallel *n*-level Coarsening

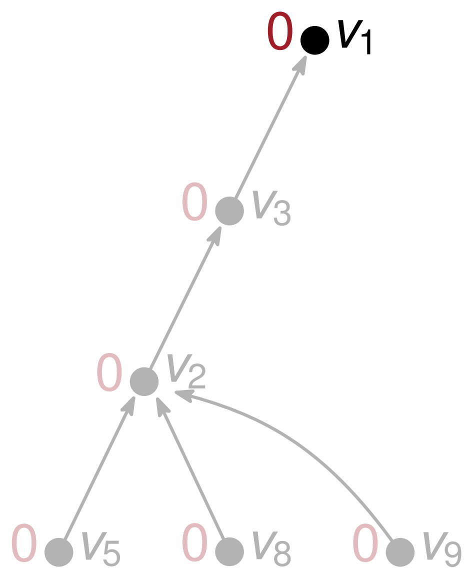
for each $u \in V$ **in parallel**
    $v \leftarrow$ find contraction partner for u
    if *add* (v, u) *to contraction forest* **then**
        contract v onto u

Contraction Forest Construction

Karlsruhe Institute of Technology

Idea: Construct contraction forest *on-the-fly*

T_i = Thead i



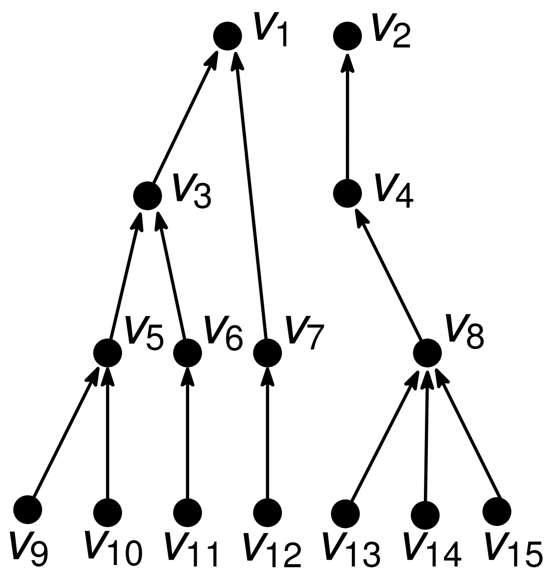
Algorithm 1: Parallel *n*-level Coarsening

for each $u \in V$ **in parallel**
  $v \leftarrow$ find contraction partner for u
  if *add* (v, u) *to contraction forest* **then**
    contract v onto u

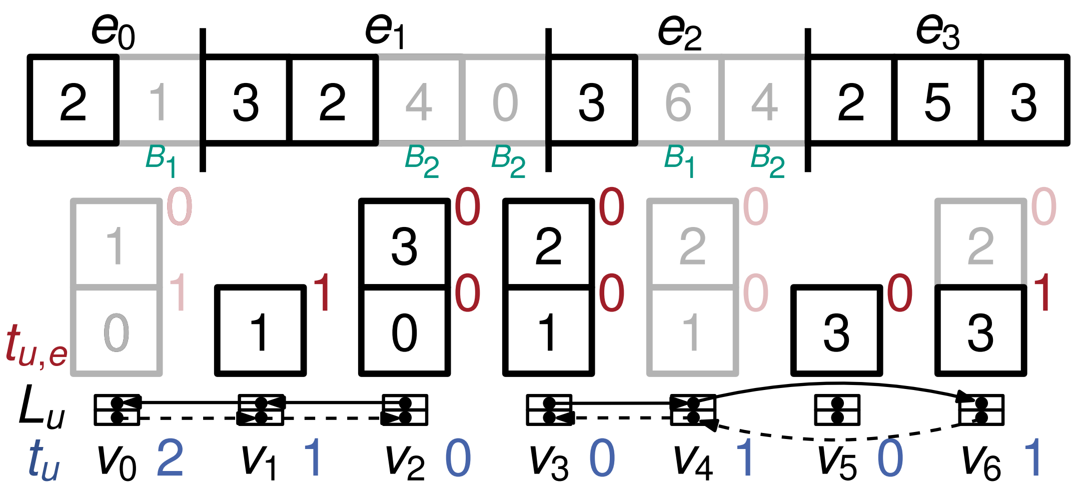
- Simple locking protocol used to modify contraction forest

Consistency Requirements

Contraction Consistency

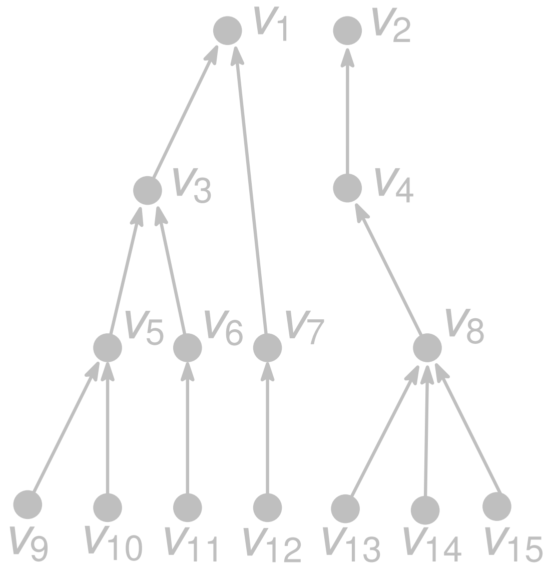


Data Structure Consistency

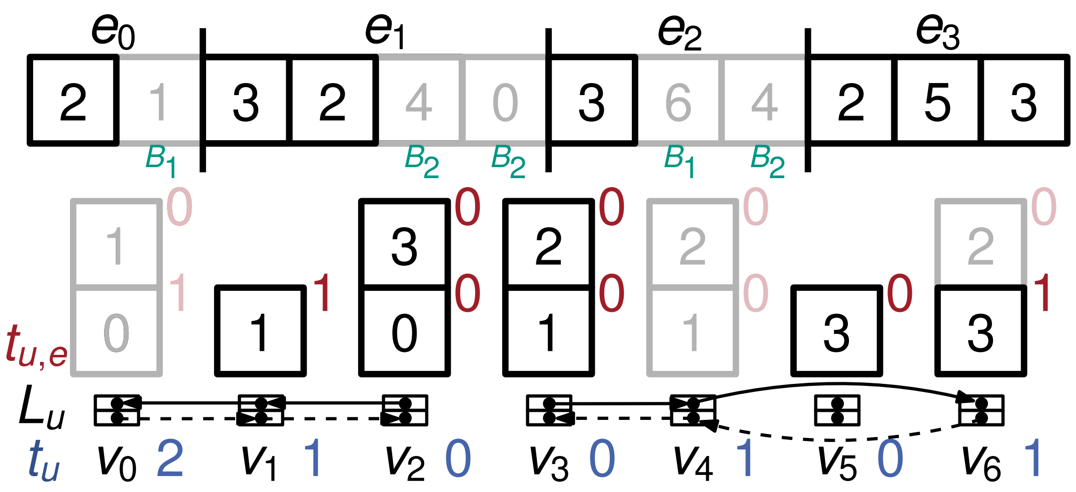


Consistency Requirements

Contraction Consistency



Data Structure Consistency



see paper

Parallel Uncoarsening

- traditional *n*-level uncontracts only **one** vertex on each level $\Rightarrow$ inherently sequential

Parallel Uncoarsening

- traditional n-level uncontracts only **one** vertex on each level $\Rightarrow$ inherently sequential

Idea

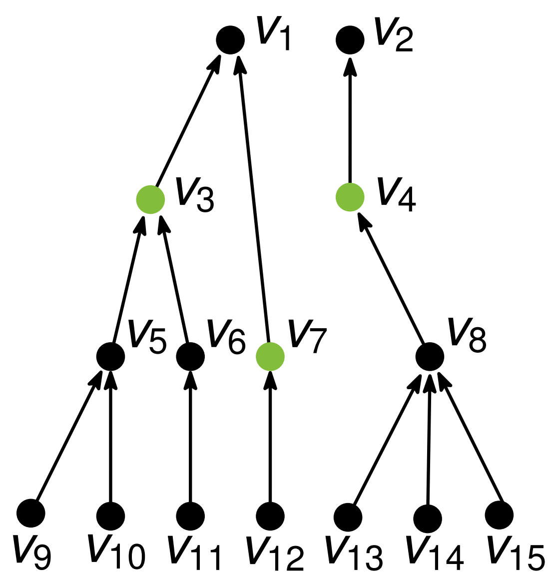
- assemble independent uncontractions in a *batch* B with $|B| \approx b_{max}$
  - uncontract B in parallel
  - then run parallel localized refinement around B
- construct *batches* $\mathcal{B} = \langle B_1, \ldots, B_l \rangle$
- uncontracting B_i enables uncontraction of all vertices in B_{i+1}

Parallel Uncoarsening

- traditional n-level uncontracts only **one** vertex on each level $\Rightarrow$ inherently sequential

Idea

- assemble independent uncontractions in a *batch* B with $|B| \approx b_{max}$
  - uncontract B in parallel
  - then run parallel localized refinement around B
- construct *batches* $\mathcal{B} = \langle B_1, \ldots, B_l \rangle$
- uncontracting B_i enables uncontraction of all vertices in B_{i+1}
- **top-down traversal** of contraction forest $\mathcal{F}$



eligible for uncontraction
already uncontracted

$b_{max} = 3$

$\mathcal{B} = \langle \square\square\square, \square\square\square, \square\square\square, \square\square\square, \square\square\square \rangle$

Parallel Uncoarsening

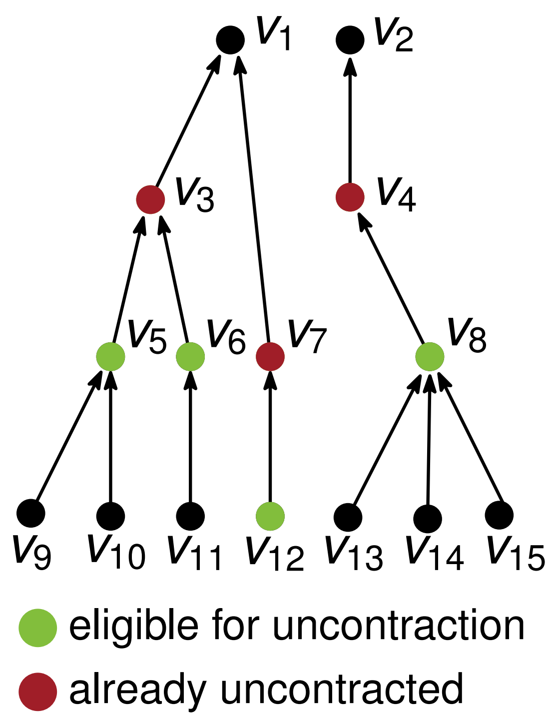
- traditional n-level uncontracts only **one** vertex on each level $\Rightarrow$ inherently sequential

Idea

- assemble independent uncontractions in a *batch* B with $|B| \approx b_{max}$
  - uncontract B in parallel
  - then run parallel localized refinement around B
- construct *batches* $\mathcal{B} = \langle B_1, \ldots, B_l \rangle$
- uncontracting B_i enables uncontraction of all vertices in B_{i+1}
- **top-down traversal** of contraction forest $\mathcal{F}$

$b_{max} = 3$

$\mathcal{B} = \langle$ [v_3 | v_7 | v_4], [| |], [| |], [| |], [| |] $\rangle$



Parallel Uncoarsening

Karlsruhe Institute of Technology

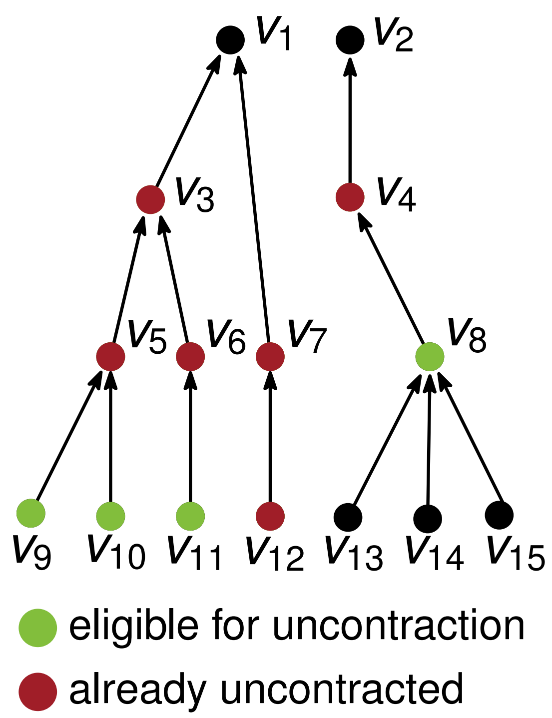
- traditional n-level uncontracts only **one** vertex on each level $\Rightarrow$ inherently sequential

Idea

- assemble independent uncontractions in a *batch* B with $|B| \approx b_{max}$
  - uncontract B in parallel
  - then run parallel localized refinement around B
- construct *batches* $\mathcal{B} = \langle B_1, \ldots, B_l \rangle$
- uncontracting B_i enables uncontraction of all vertices in B_{i+1}
- **top-down traversal** of contraction forest $\mathcal{F}$

$b_{max} = 3$

$\mathcal{B} = \langle \boxed{v_3 \mid v_7 \mid v_4}, \boxed{v_5 \mid v_6 \mid v_{12}}, \boxed{\quad \mid \quad \mid \quad}, \boxed{\quad \mid \quad \mid \quad}, \boxed{\quad \mid \quad \mid \quad} \rangle$



Parallel Uncoarsening

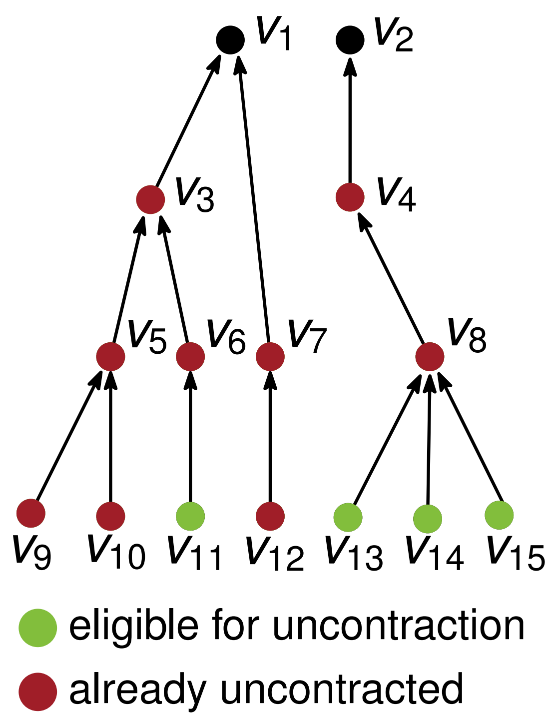
- traditional n-level uncontracts only **one** vertex on each level $\Rightarrow$ inherently sequential

Idea

- assemble independent uncontractions in a *batch* B with $|B| \approx b_{max}$
  - uncontract B in parallel
  - then run parallel localized refinement around B
- construct *batches* $\mathcal{B} = \langle B_1, \ldots, B_l \rangle$
- uncontracting B_i enables uncontraction of all vertices in B_{i+1}
- **top-down traversal** of contraction forest $\mathcal{F}$

$b_{max} = 3$

$$\mathcal{B} = \langle \boxed{v_3 \mid v_7 \mid v_4}, \boxed{v_5 \mid v_6 \mid v_{12}}, \boxed{v_8 \mid v_9 \mid v_{10}}, \boxed{\quad \mid \quad \mid \quad}, \boxed{\quad \mid \quad \mid \quad} \rangle$$



Parallel Uncoarsening

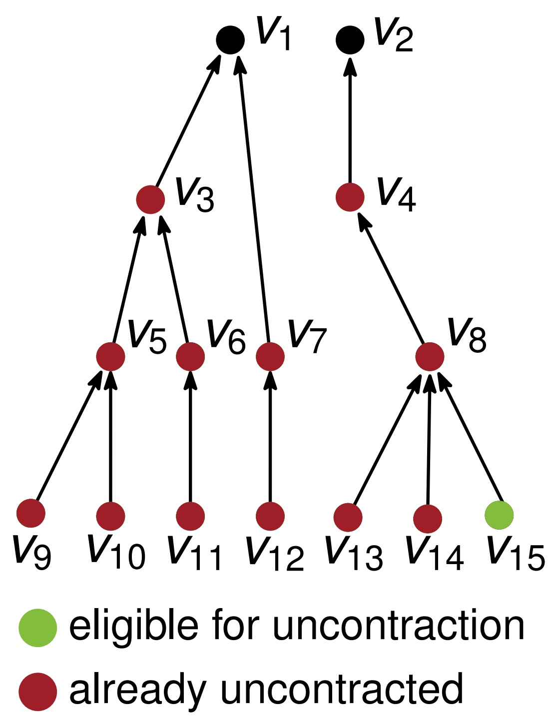
- traditional n-level uncontracts only **one** vertex on each level $\Rightarrow$ inherently sequential

Idea

- assemble independent uncontractions in a *batch* B with $|B| \approx b_{max}$
  - uncontract B in parallel
  - then run parallel localized refinement around B
- construct *batches* $\mathcal{B} = \langle B_1, \ldots, B_l \rangle$
- uncontracting B_i enables uncontraction of all vertices in B_{i+1}
- **top-down traversal** of contraction forest $\mathcal{F}$

$b_{max} = 3$

$\mathcal{B} = \langle [v_3 \mid v_7 \mid v_4], [v_5 \mid v_6 \mid v_{12}], [v_8 \mid v_9 \mid v_{10}], [v_{11} \mid v_{13} \mid v_{14}], [\ \mid\ \mid\ ] \rangle$



Parallel Uncoarsening

Karlsruhe Institute of Technology

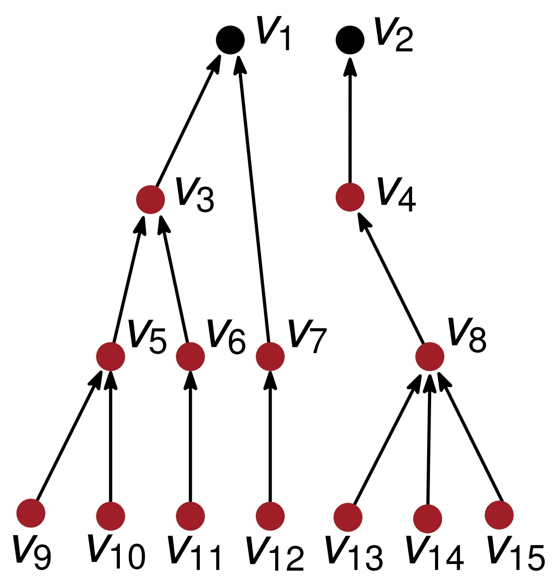
- traditional n-level uncontracts only **one** vertex on each level $\Rightarrow$ inherently sequential

Idea

- assemble independent uncontractions in a *batch* B with $|B| \approx b_{max}$
  - uncontract B in parallel
  - then run parallel localized refinement around B
- construct *batches* $\mathcal{B} = \langle B_1, \ldots, B_l \rangle$
- uncontracting B_i enables uncontraction of all vertices in B_{i+1}
- **top-down traversal** of contraction forest $\mathcal{F}$

$b_{max} = 3$

$\mathcal{B} = \langle [v_3 \mid v_7 \mid v_4], [v_5 \mid v_6 \mid v_{12}], [v_8 \mid v_9 \mid v_{10}], [v_{11} \mid v_{13} \mid v_{14}], [v_{15} \mid \ \mid \ ] \rangle$



- eligible for uncontraction
- already uncontracted

Parallel Uncoarsening

Karlsruhe Institute of Technology

- traditional *n*-level uncontracts only **one** vertex on each level $\Rightarrow$ inherently sequential

Idea

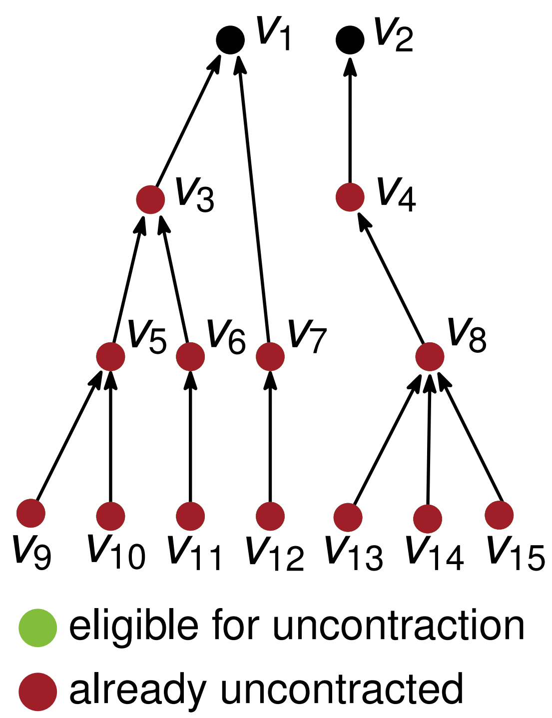
- assemble independent uncontractions in a *batch* B with $|B| \approx b_{max}$
  - uncontract B in parallel
  - then run parallel localized refinement around B

b_{max} = 1000 in practice

- construct *batches* $\mathcal{B} = \langle B_1, \ldots, B_l \rangle$
- uncontracting B_i enables uncontraction of all vertices in B_{i+1}
- **top-down traversal** of contraction forest $\mathcal{F}$

$b_{max} = 3$

$\mathcal{B} = \langle [v_3 \mid v_7 \mid v_4], [v_5 \mid v_6 \mid v_{12}], [v_8 \mid v_9 \mid v_{10}], [v_{11} \mid v_{13} \mid v_{14}], [v_{15} \mid \; \mid \;] \rangle$



Parallel Uncoarsening

- traditional n-level uncontracts only **one** vertex on each level $\Rightarrow$ inherently sequential

Idea

- assemble independent uncontractions in a *batch* B with $|B| \approx b_{max}$
  - uncontract B in parallel
  - then run parallel localized refinement around B

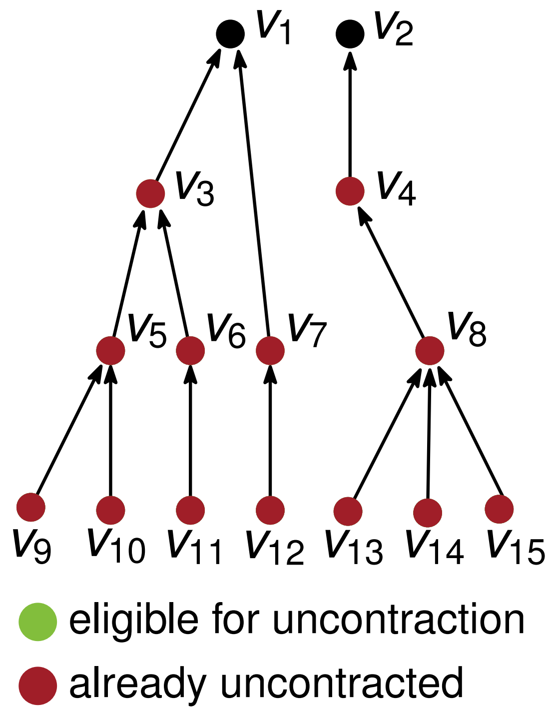
$b_{max} = 1000$ in practice

- construct *batches* $\mathcal{B} = \langle B_1, \ldots, B_k \rangle$
- uncontracting B_i enables uncontraction of all vertices in B_{i+1}
- **top-down traversal** of contraction forest $\mathcal{F}$

Implementation Detail:
Uncontract siblings in reverse order of contraction
$\Rightarrow$ *see paper*

$b_{max} = 3$

$\mathcal{B} = \langle [v_3 \mid v_7 \mid v_4], [v_5 \mid v_6 \mid v_{12}], [v_8 \mid v_9 \mid v_{10}], [v_{11} \mid v_{13} \mid v_{14}], [v_{15} \mid \ \mid \ ] \rangle$

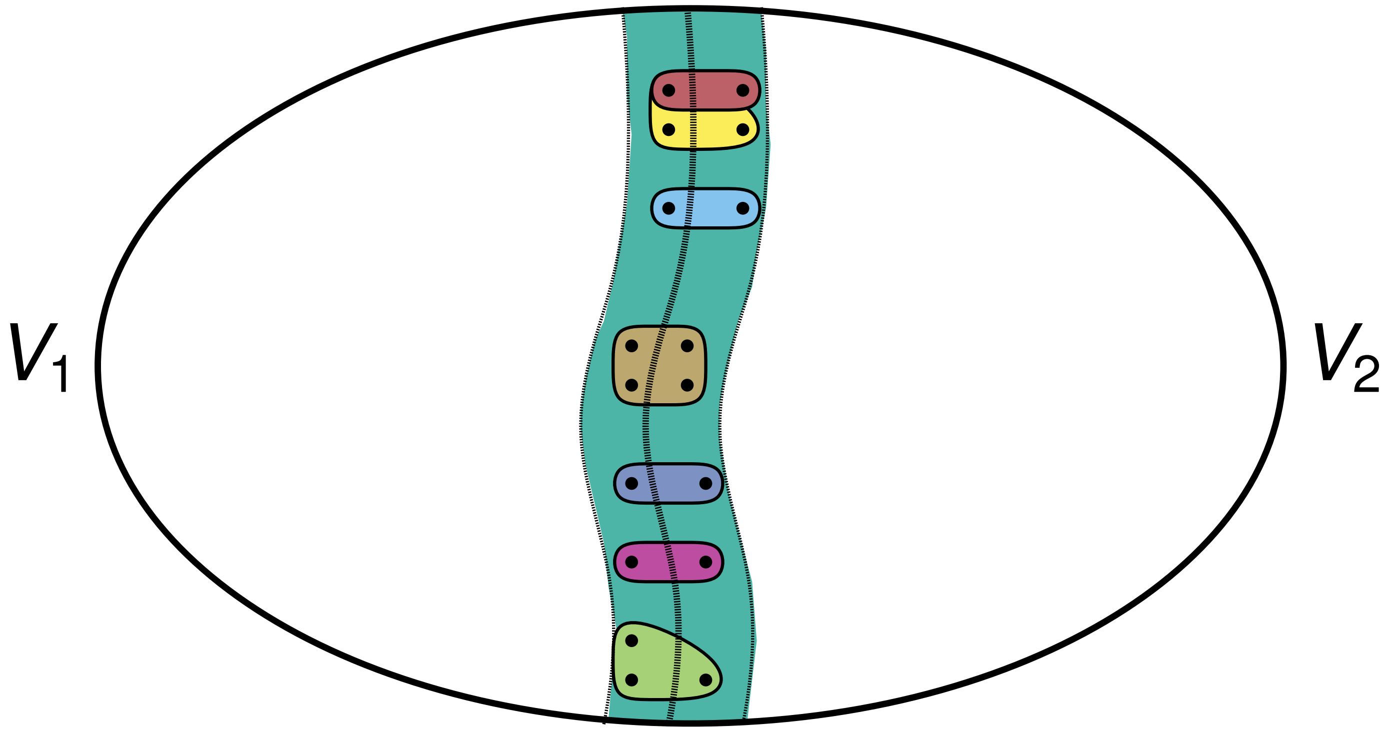


Parallel Flow-Based Refinement

The value of a **maxium flow** between to vertices *s* and *t* is equal with the **minimum cut** seperating *s* and *t*

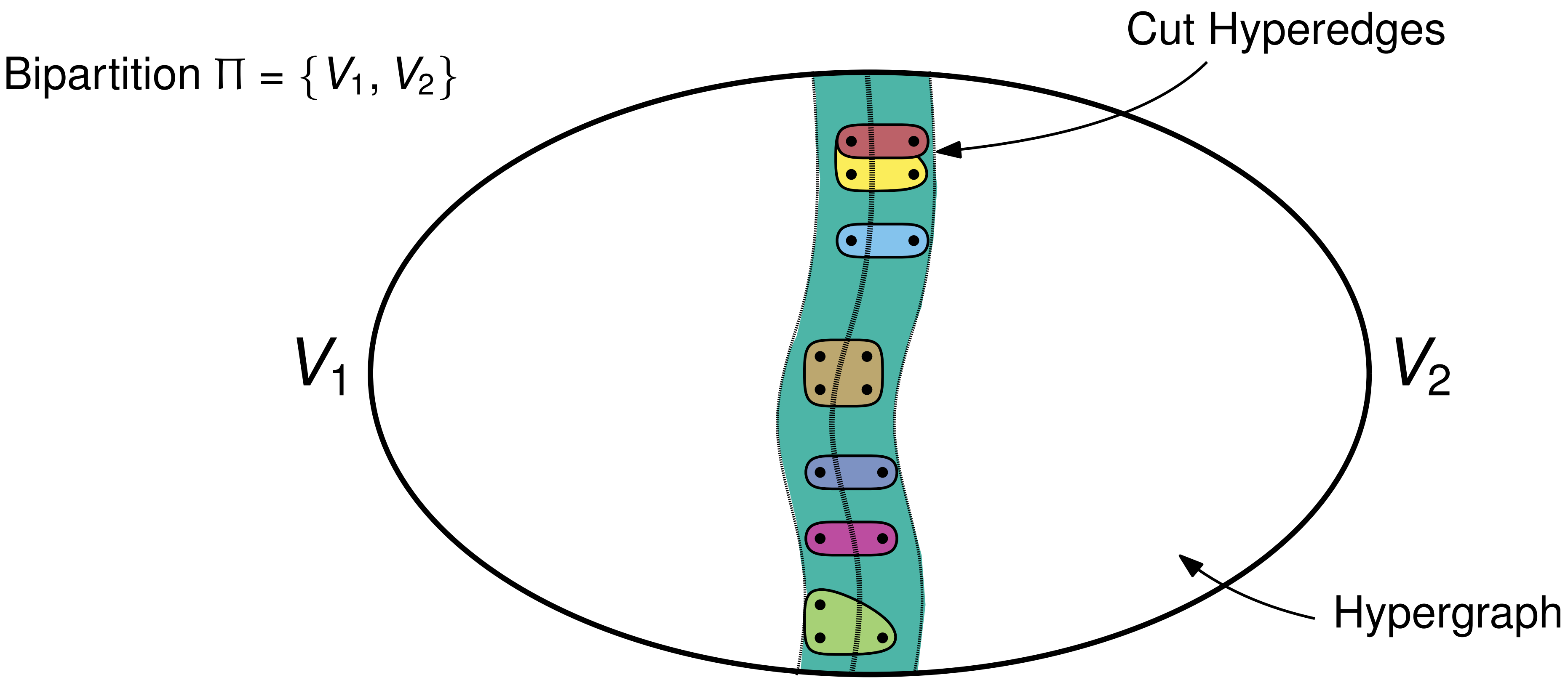
Parallel Flow-Based Refinement

The value of a **maxium flow** between to vertices *s* and *t* is equal with the **minimum cut** seperating *s* and *t*



Parallel Flow-Based Refinement

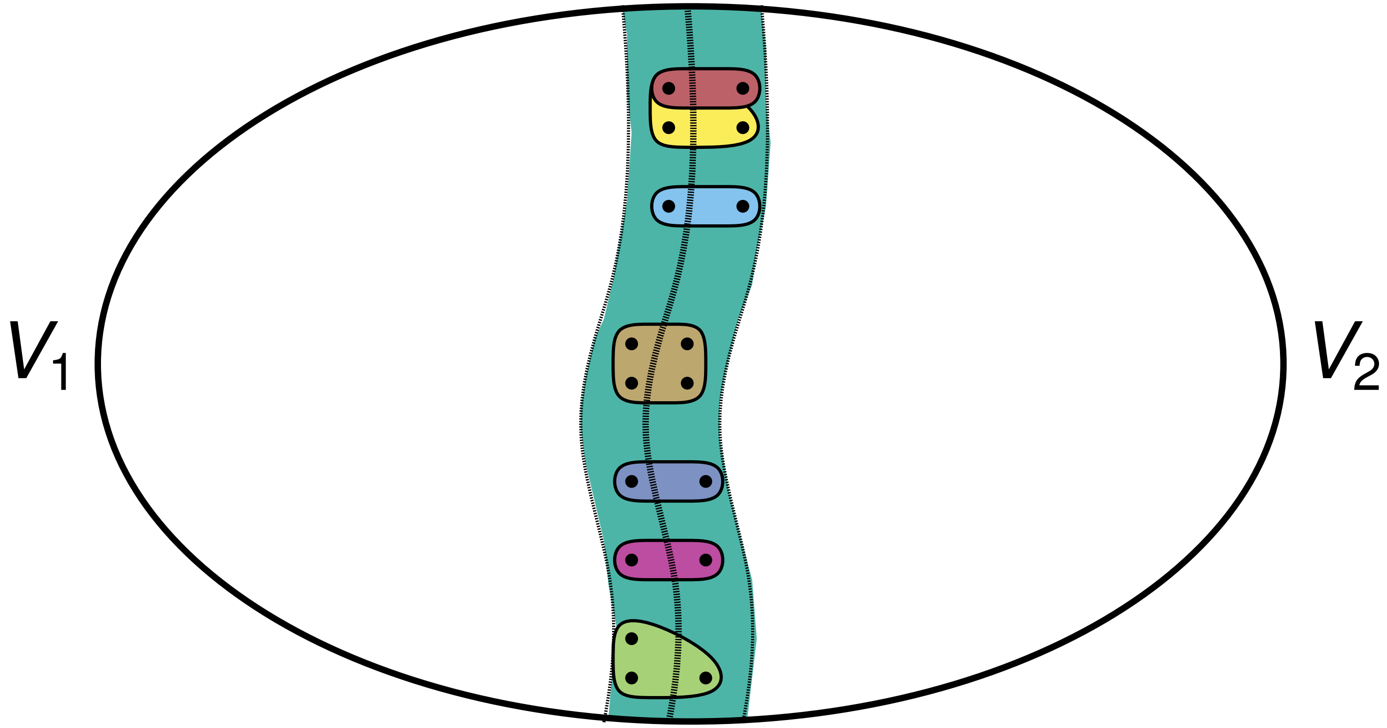
The value of a **maxium flow** between to vertices *s* and *t* is equal with the **minimum cut** seperating *s* and *t*



Parallel Flow-Based Refinement

The value of a **maxium flow** between to vertices *s* and *t* is equal with the **minimum cut** seperating *s* and *t*

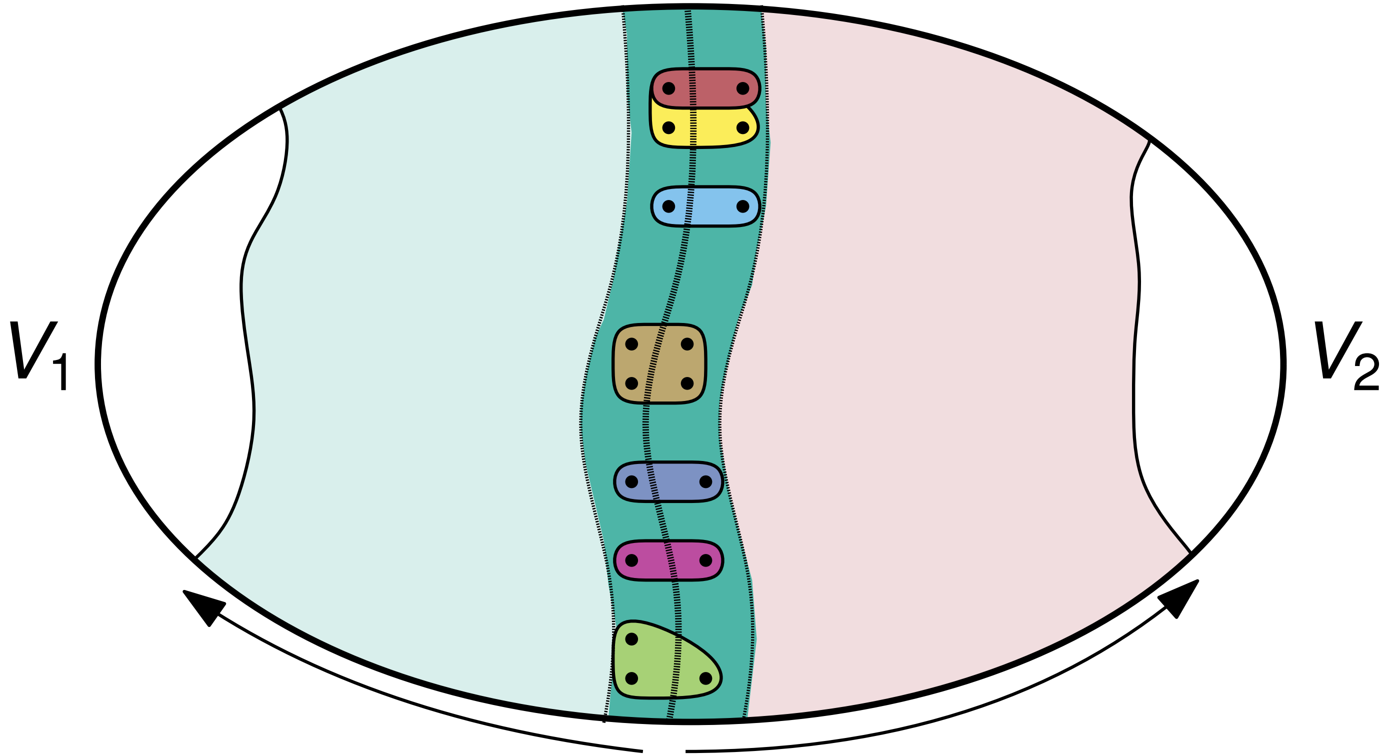
Initial Cut = 539, Target Imbalance = 3%



Parallel Flow-Based Refinement

The value of a **maxium flow** between to vertices *s* and *t* is equal with the **minimum cut** seperating *s* and *t*

Initial Cut = 539, Target Imbalance = 3%



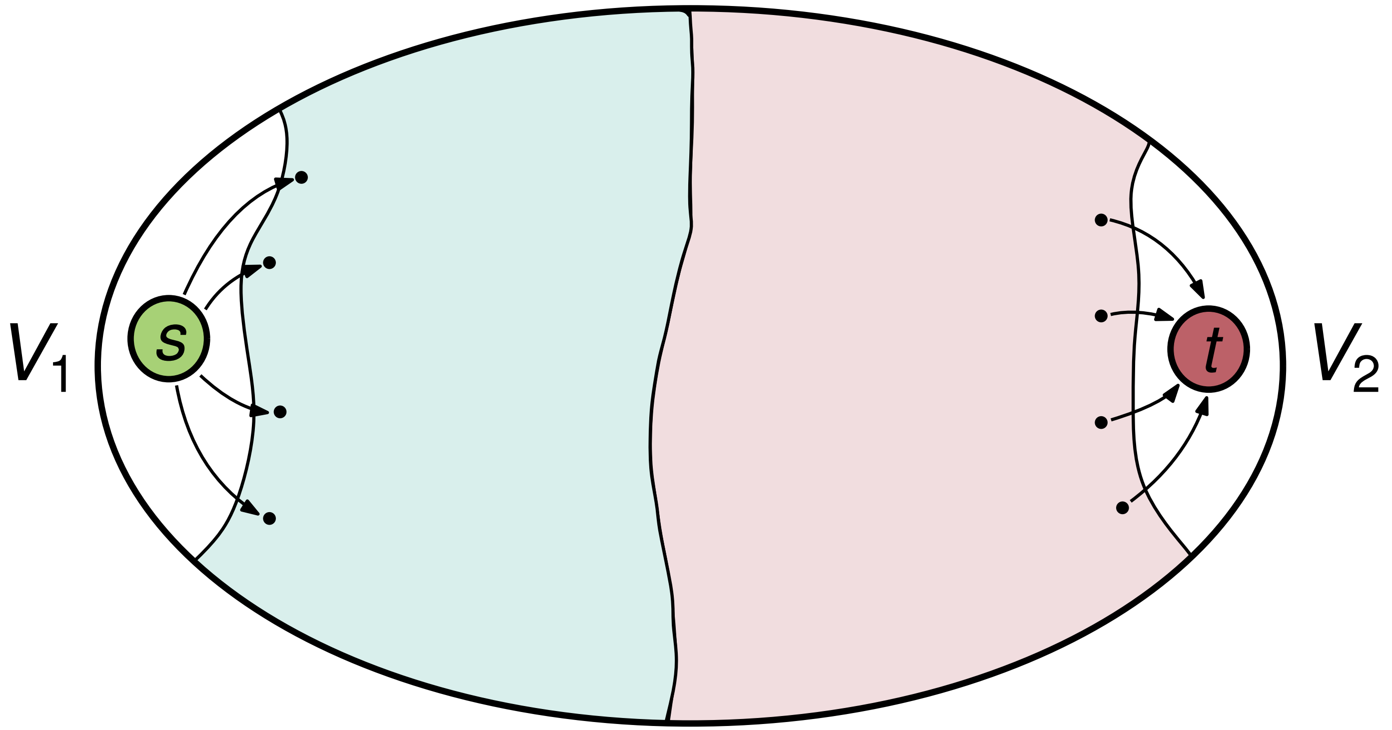
Grow region around cut via BFS

Parallel Flow-Based Refinement

Karlsruhe Institute of Technology

The value of a **maxium flow** between to vertices *s* and *t* is equal with the **minimum cut** seperating *s* and *t*

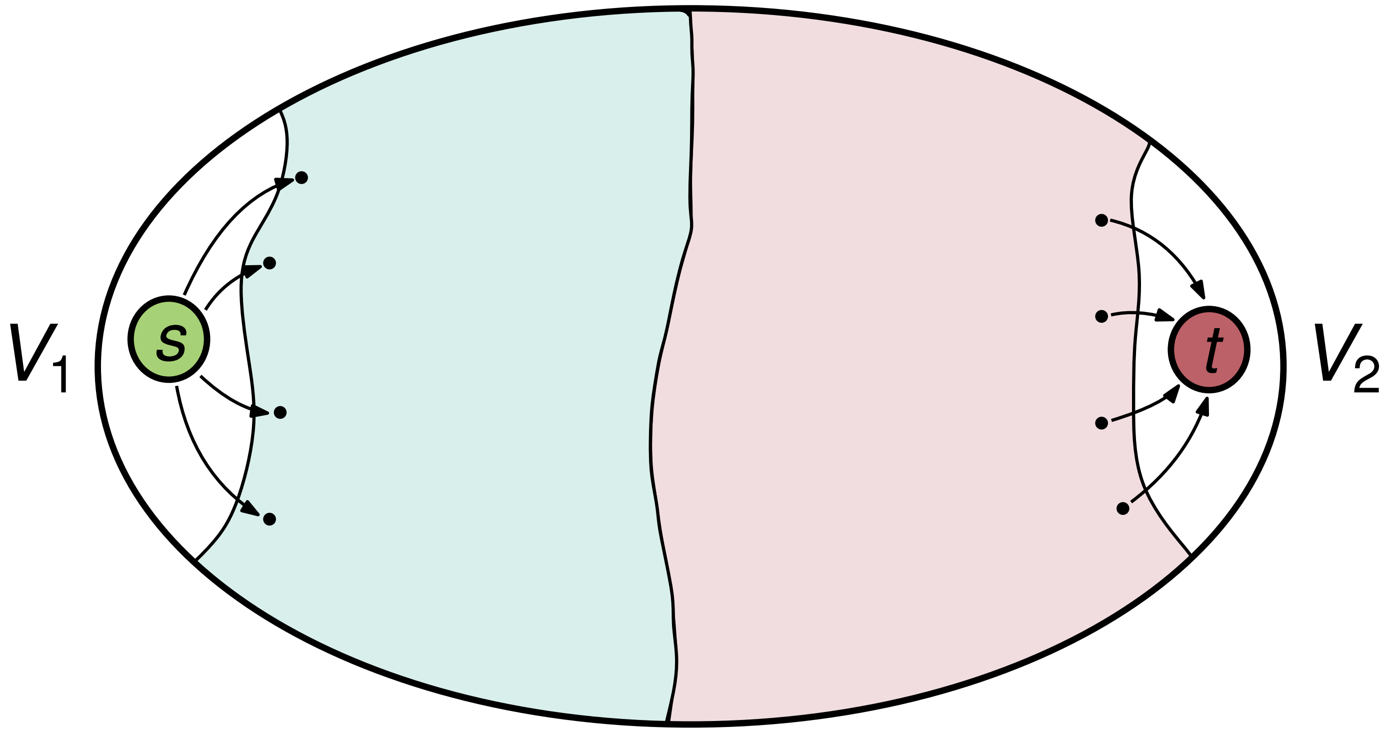
Initial Cut = 539, Target Imbalance = 3%



Parallel Flow-Based Refinement

The value of a **maxium flow** between to vertices *s* and *t* is equal with the **minimum cut** seperating *s* and *t*

Initial Cut = 539, Target Imbalance = 3%

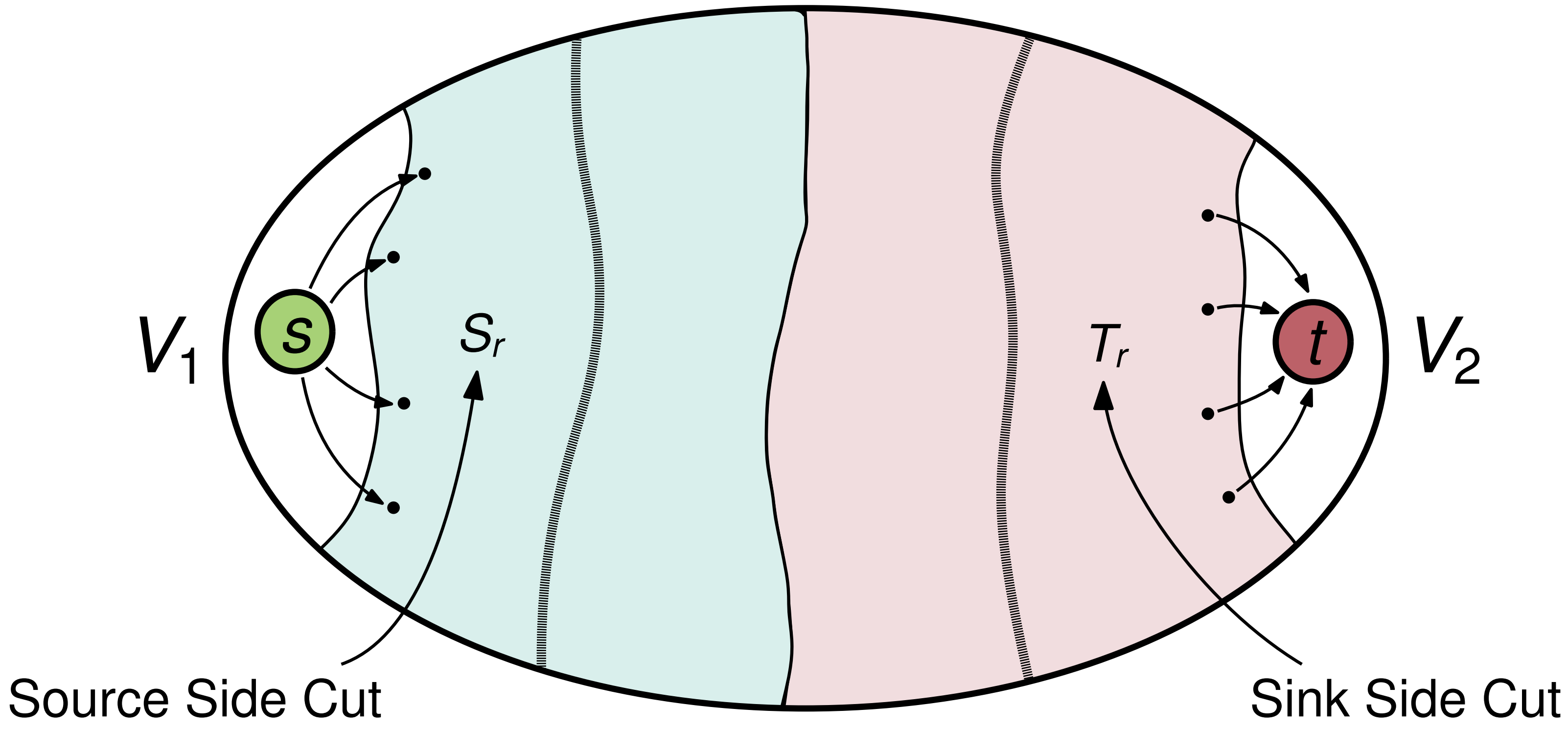


Compute a maximum (*s*, *t*)-flow

Parallel Flow-Based Refinement

The value of a **maxium flow** between to vertices *s* and *t* is equal with the **minimum cut** seperating *s* and *t*

Initial Cut = 539, Target Imbalance = 3%



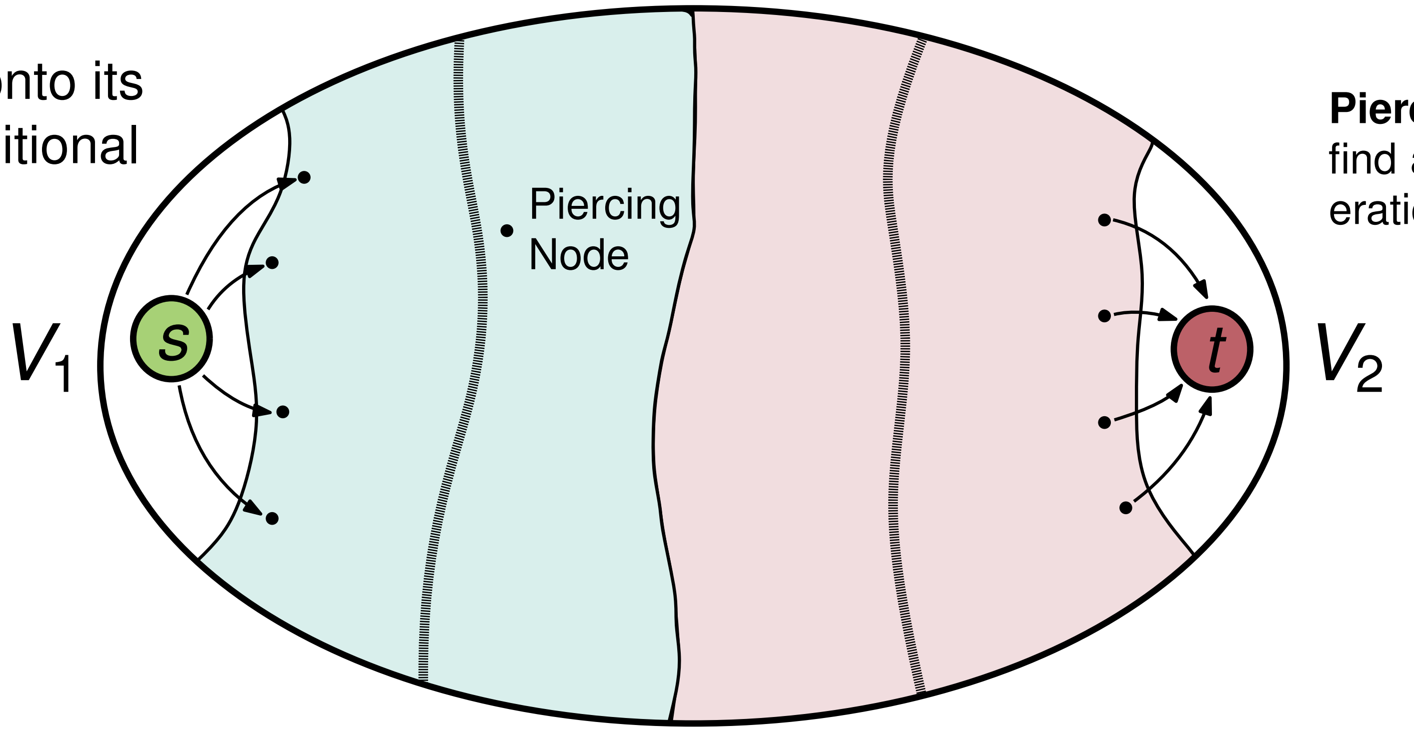
Current Cut = 250, Current Imbalance = 15% **Imbalanced!**

Parallel Flow-Based Refinement

The value of a **maxium flow** between to vertices *s* and *t* is equal with the **minimum cut** seperating *s* and *t*

Initial Cut = 539, Target Imbalance = 3%

Contract smaller cut onto its terminal plus one additional node



Piercing node ensure that we find a different cut in the next iteration

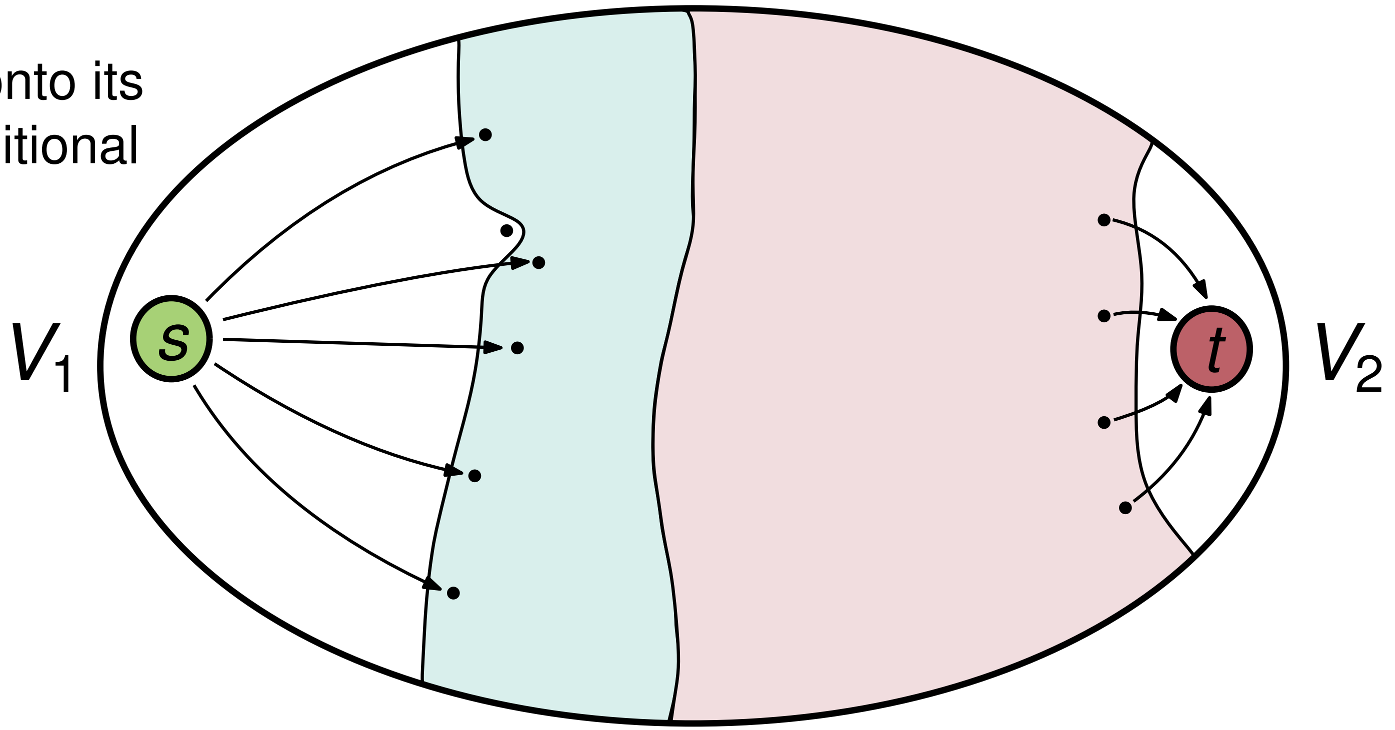
Current Cut = 250, Current Imbalance = 15% **Imbalanced!**

Parallel Flow-Based Refinement

The value of a **maxium flow** between to vertices *s* and *t* is equal with the **minimum cut** seperating *s* and *t*

Initial Cut = 539, Target Imbalance = 3%

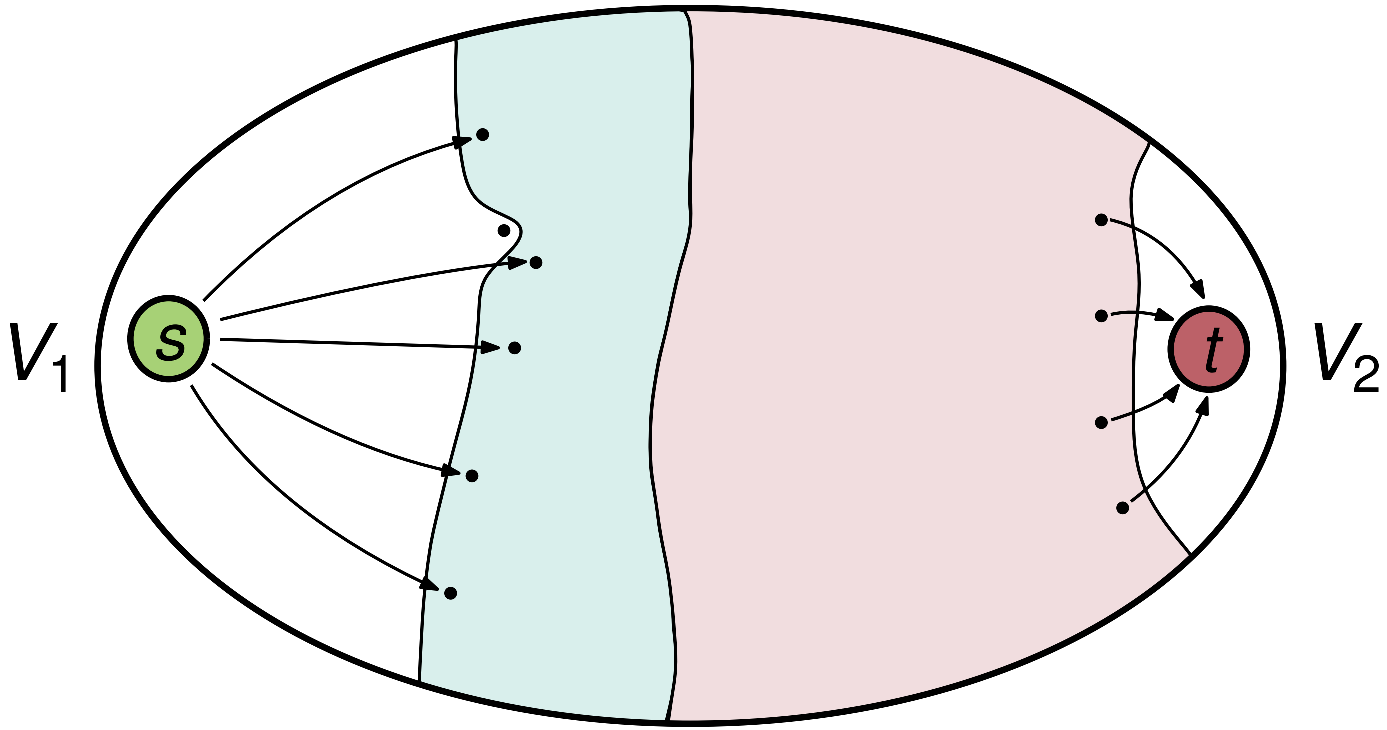
Contract smaller cut onto its terminal plus one additional node



Parallel Flow-Based Refinement

The value of a **maxium flow** between to vertices *s* and *t* is equal with the **minimum cut** seperating *s* and *t*

Initial Cut = 539, Target Imbalance = 3%

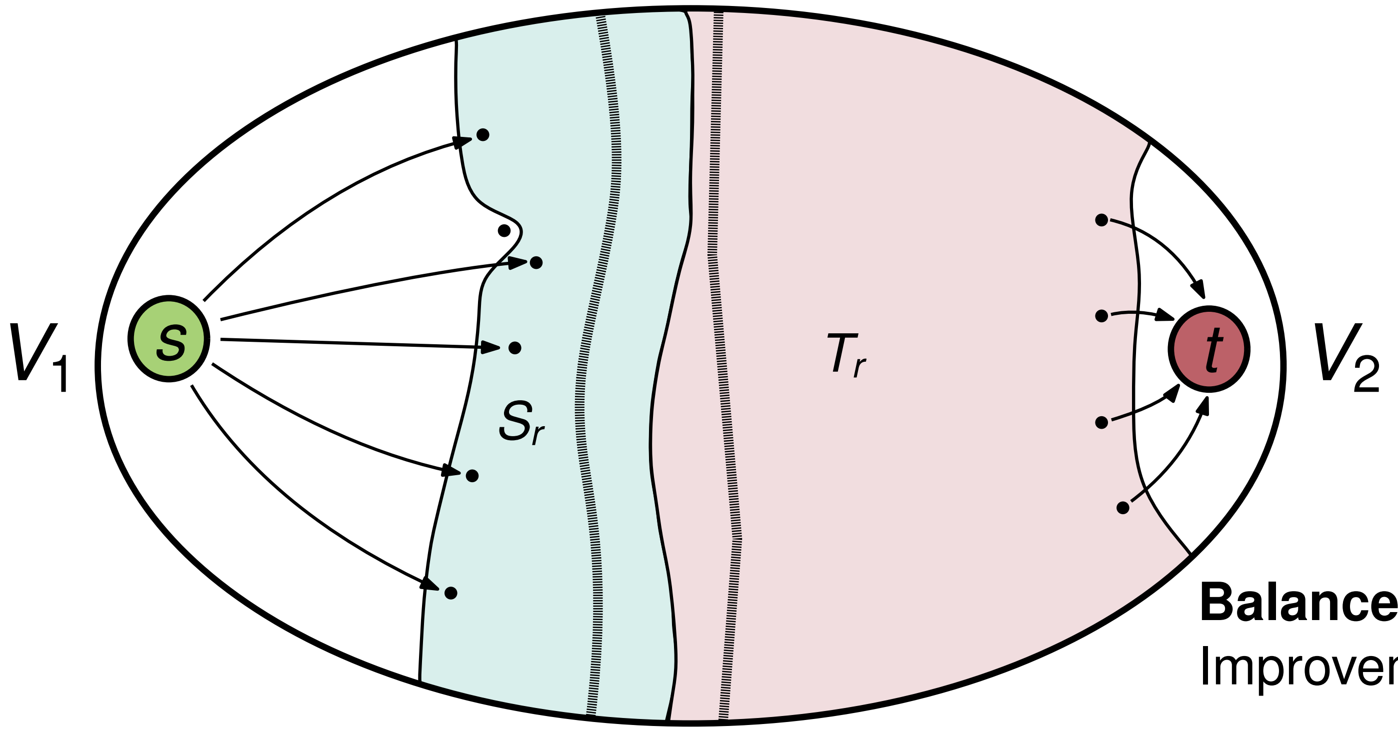


Augment flow again to a maximum (*s*, *t*)-flow

Parallel Flow-Based Refinement

The value of a **maxium flow** between to vertices *s* and *t* is equal with the **minimum cut** seperating *s* and *t*

Initial Cut = 539, Target Imbalance = 3%



Balanced!
Improvement = 539 − 498 = 41

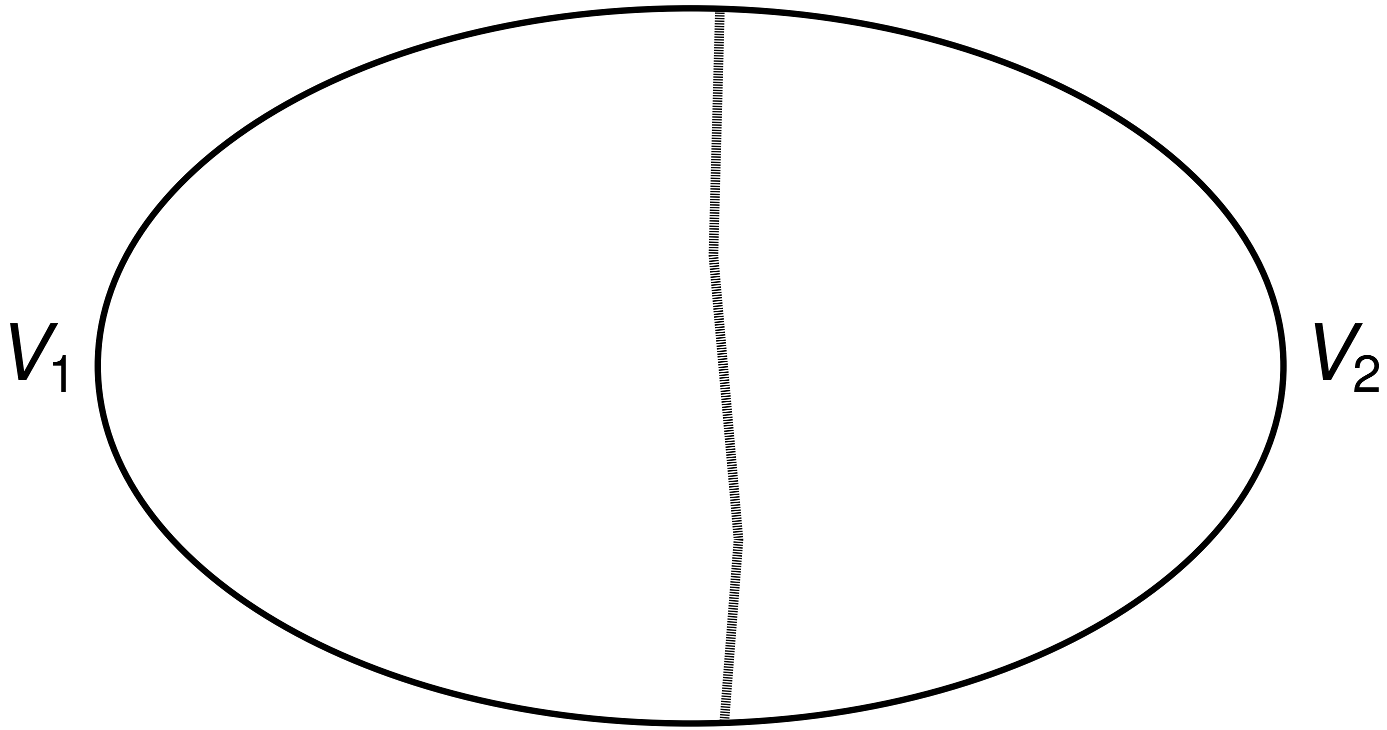
Current Cut = 498, Current Imbalance = 2.5%

Parallel Flow-Based Refinement

Karlsruhe Institute of Technology

The value of a **maxium flow** between to vertices *s* and *t* is equal with the **minimum cut** seperating *s* and *t*

New Cut = 498, New Imbalance = 2.5%

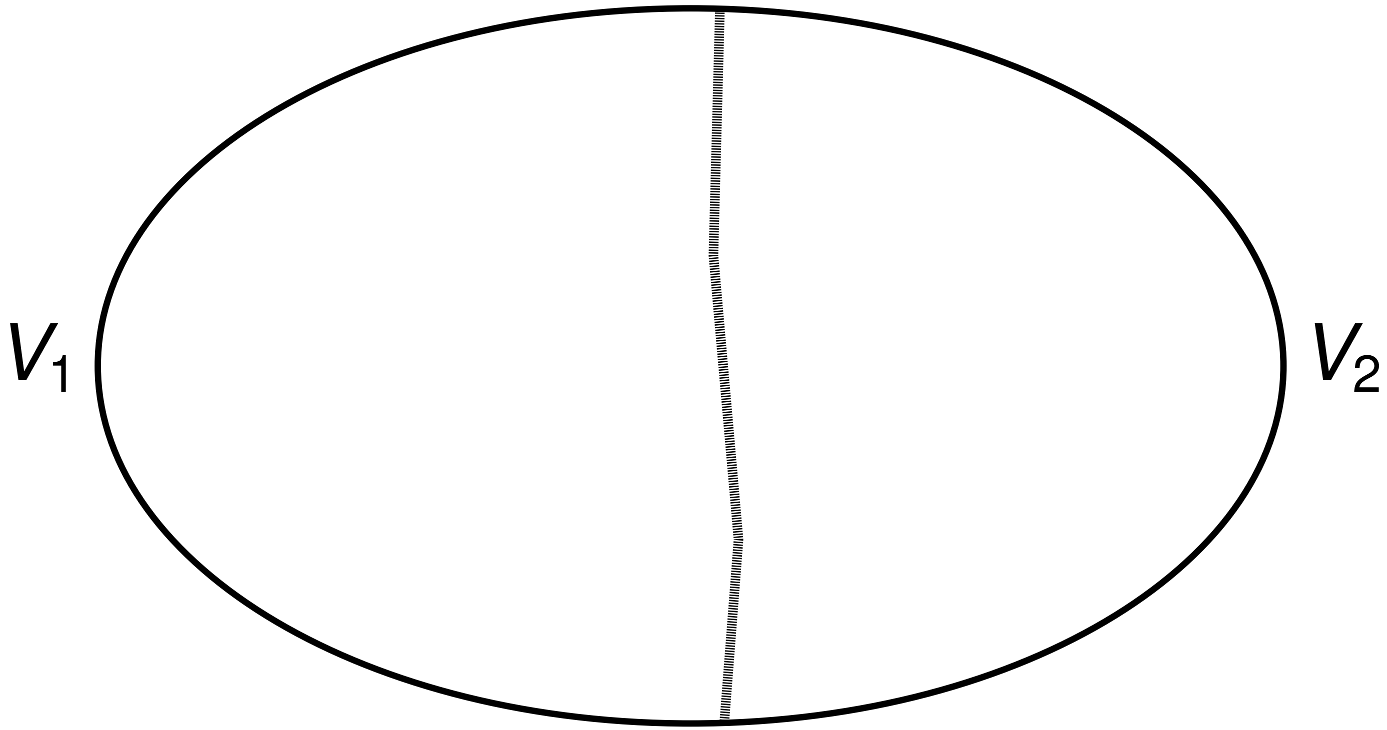


Parallel Flow-Based Refinement

Karlsruhe Institute of Technology

The value of a **maxium flow** between to vertices *s* and *t* is equal with the **minimum cut** seperating *s* and *t*

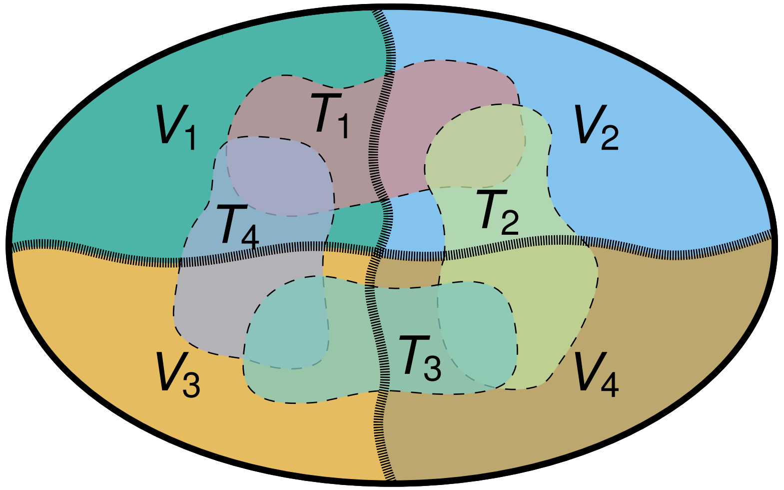
New Cut = 498, New Imbalance = 2.5%



Our implementation uses a **parallel** maximum flow algorithm (push-relabel algorithm)

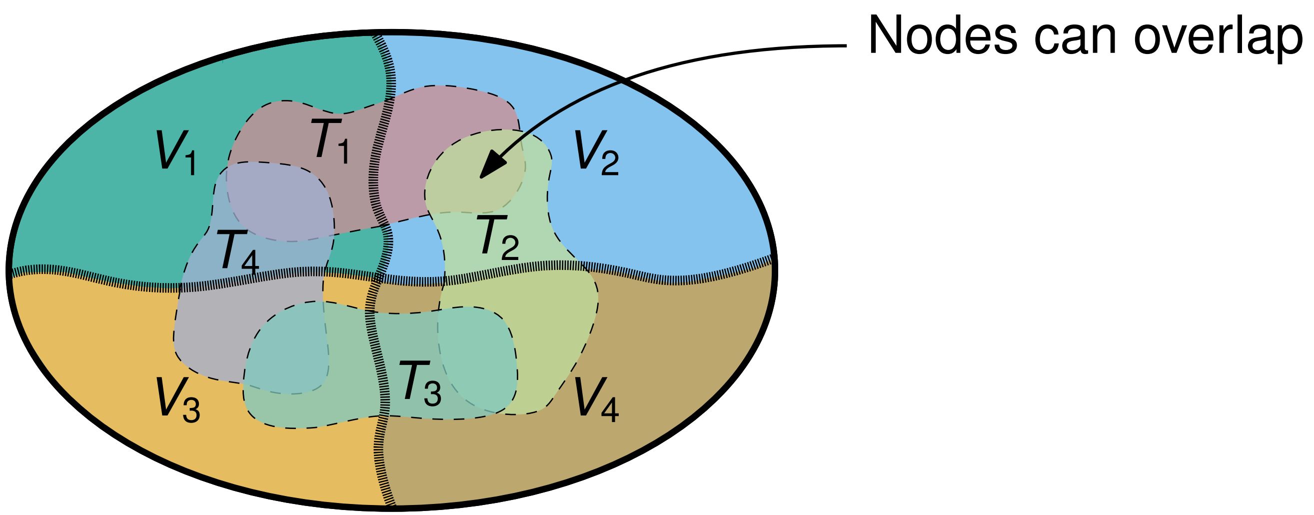
Parallel Flow-Based Refinement

General Idea: Schedule parallel flow problems on adjacent block pairs



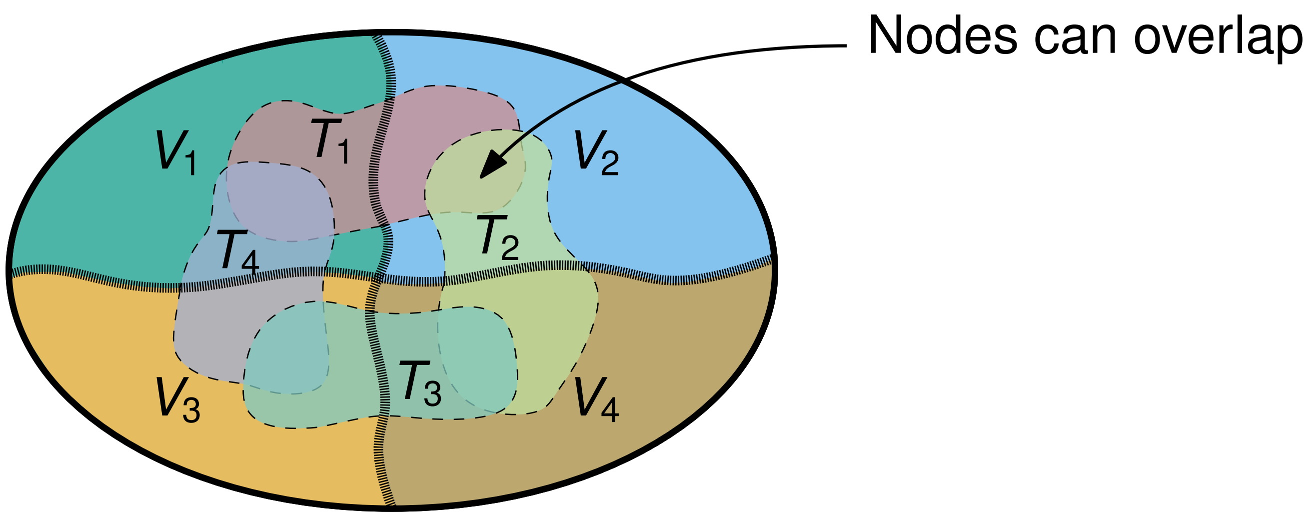
Parallel Flow-Based Refinement

General Idea: Schedule parallel flow problems on adjacent block pairs



Parallel Flow-Based Refinement

General Idea: Schedule parallel flow problems on adjacent block pairs

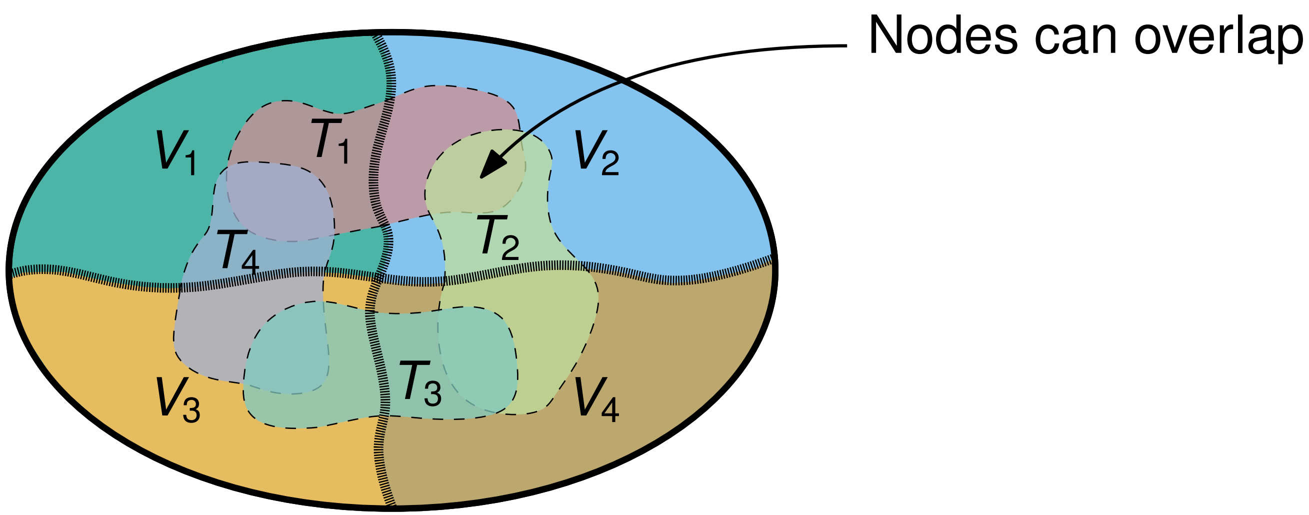


- Flow computation returns a sequences moves
- What could possibly go wrong?

Parallel Flow-Based Refinement

Karlsruhe Institute of Technology

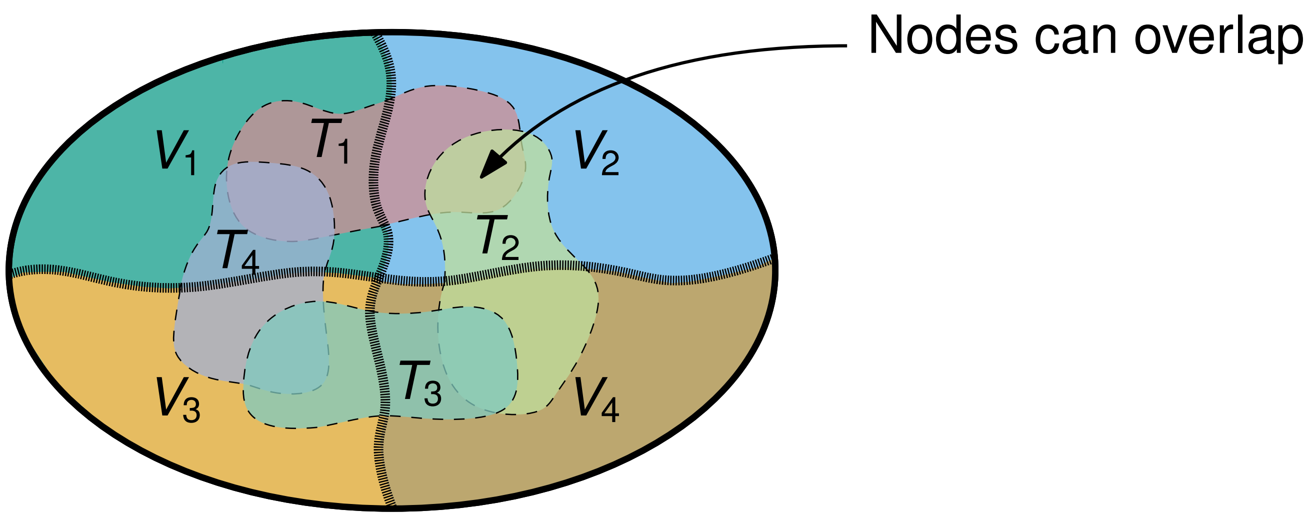
General Idea: Schedule parallel flow problems on adjacent block pairs



- Flow computation returns a sequences moves
- What could possibly go wrong?
  - Applying the move sequence could violate the balance constraint

Parallel Flow-Based Refinement

General Idea: Schedule parallel flow problems on adjacent block pairs



- Flow computation returns a sequences moves
- What could possibly go wrong?
  - Applying the move sequence could violate the balance constraint
  - Applying the move sequence could worsen the solution quality

Experiments – Large Instances

Karlsruhe Institute of Technology

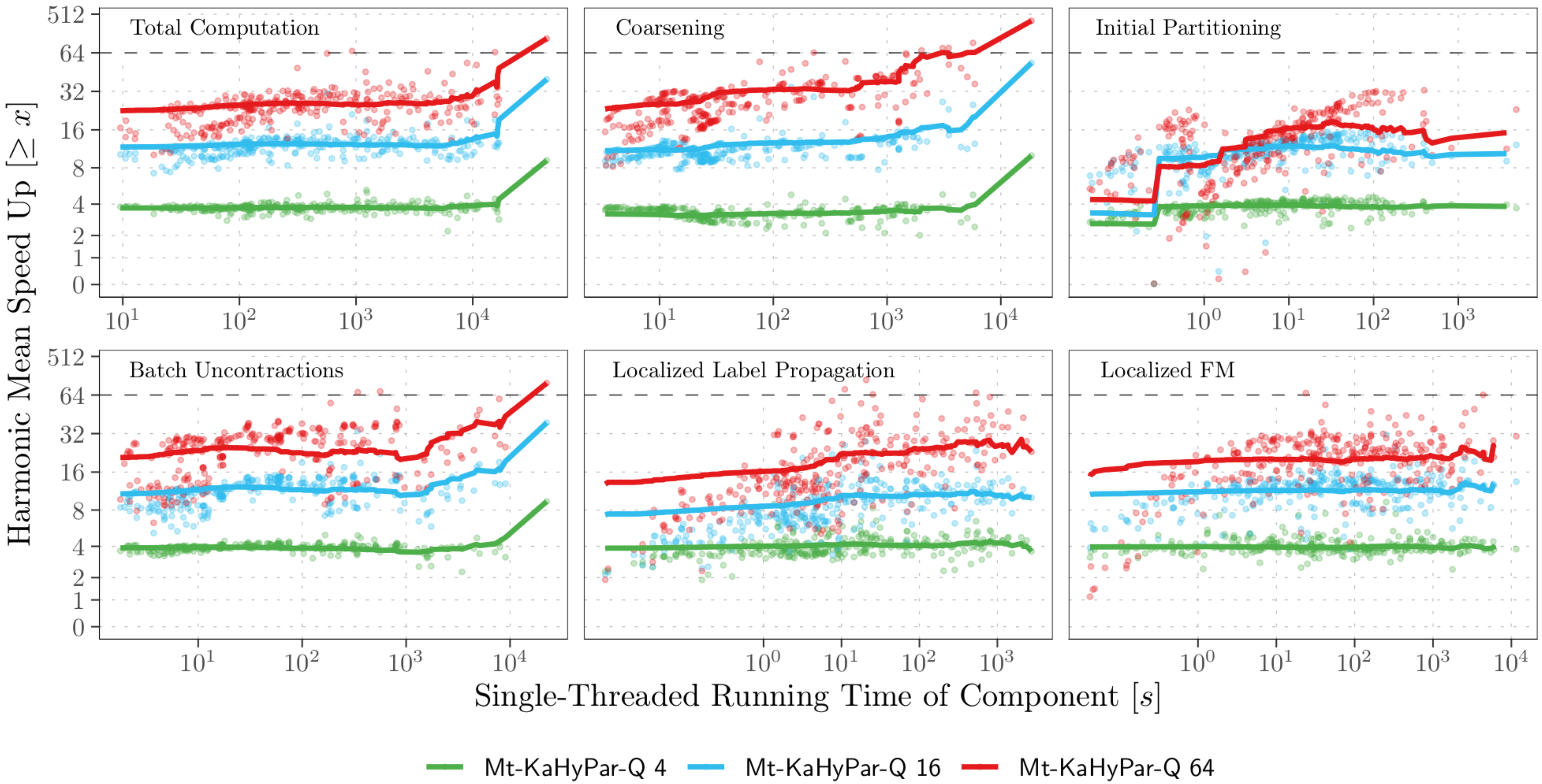
- for comparison with fast partitioners: Zoltan, PaToH-D, Hype, BiPart
- for scaling experiments

- 1st gen Epyc Rome, 1 socket, 64 cores @ 2.0-3.35 Ghz, 1024 GB RAM

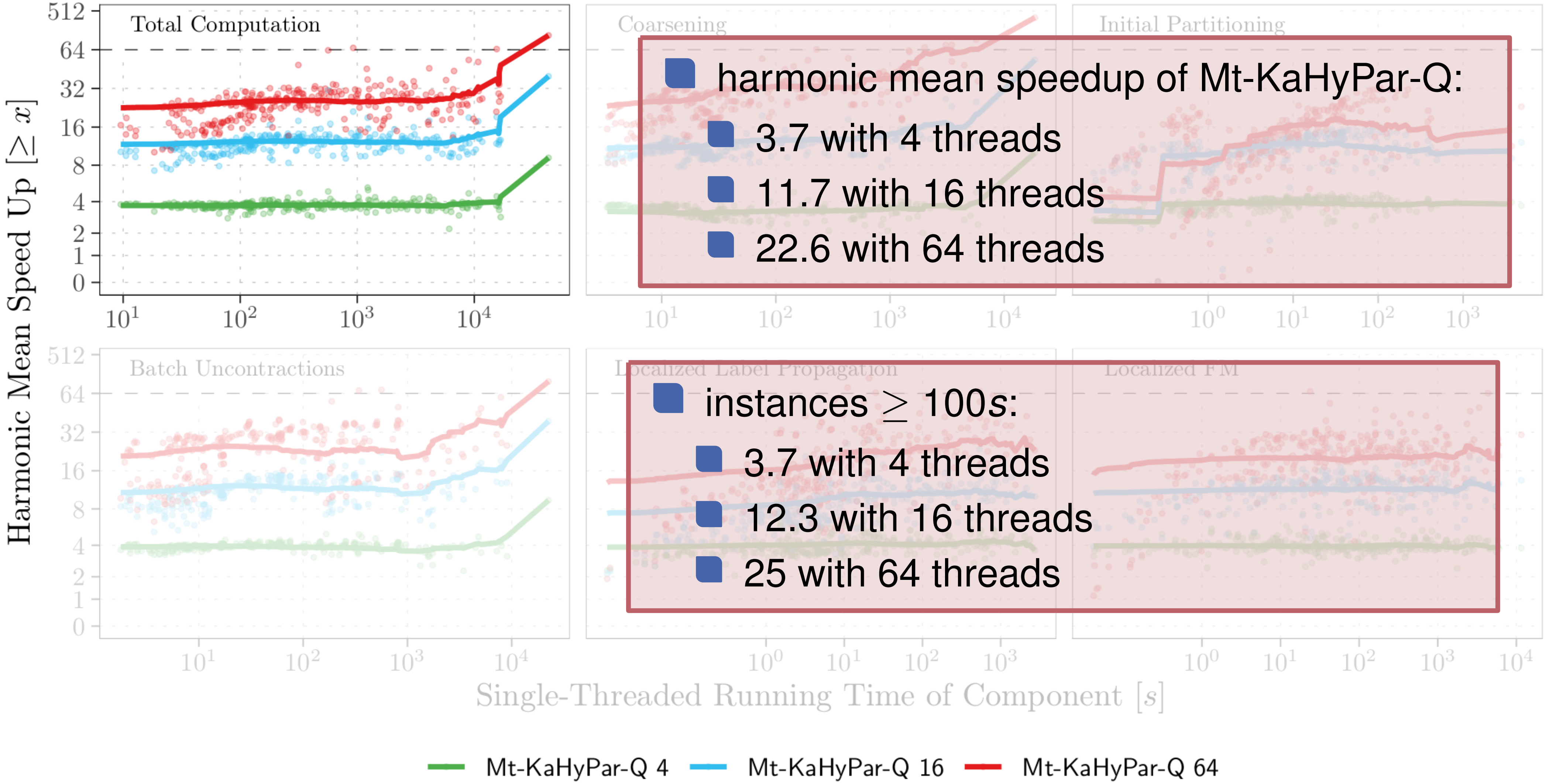
- 94 large hypergraphs: [publicly available]
  - SuiteSparse Matrix Collection — 42
  - SAT Competition 2014 (3 representations) — $14 \cdot 3 = 42$
  - DAC2012 VLSI Circuits — 10
- Largest hypergraph $\approx$ **2 billion pins**

- $k \in \{2, 8, 16, 64\}$ with imbalance: $\varepsilon = 3\%$
- 5 random seeds
- 1,4,16,64 threads

Experiments – Scalabilty



Experiments – Scalabilty



- harmonic mean speedup of Mt-KaHyPar-Q:
  - 3.7 with 4 threads
  - 11.7 with 16 threads
  - 22.6 with 64 threads

- instances $\geq 100s$:
  - 3.7 with 4 threads
  - 12.3 with 16 threads
  - 25 with 64 threads

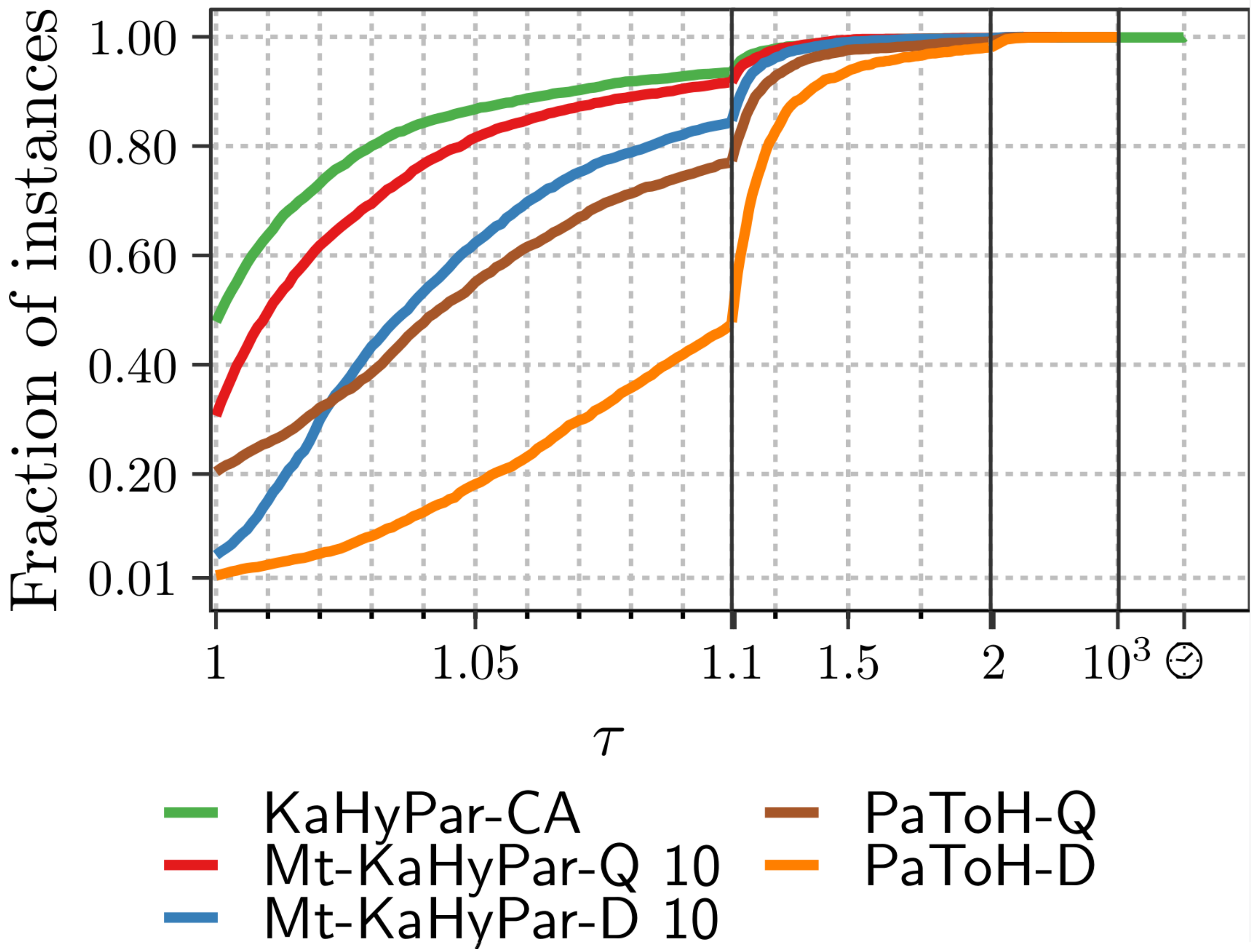
Experiments – Medium-Sized Instances

- for comparison with sequential partitioners: KaHyPar, hMetis, PaToH
- Intel Xeon Gold, 2 sockets, 20 cores @ 2.1 Ghz, 96 GB RAM

- 488 hypergraphs: [publicly available]
  - SuiteSparse Matrix Collection 184
  - SAT Competition 2014 (3 representations) 92·3 = 276
  - DAC2012 VLSI Circuits 10
  - ISPD98 18

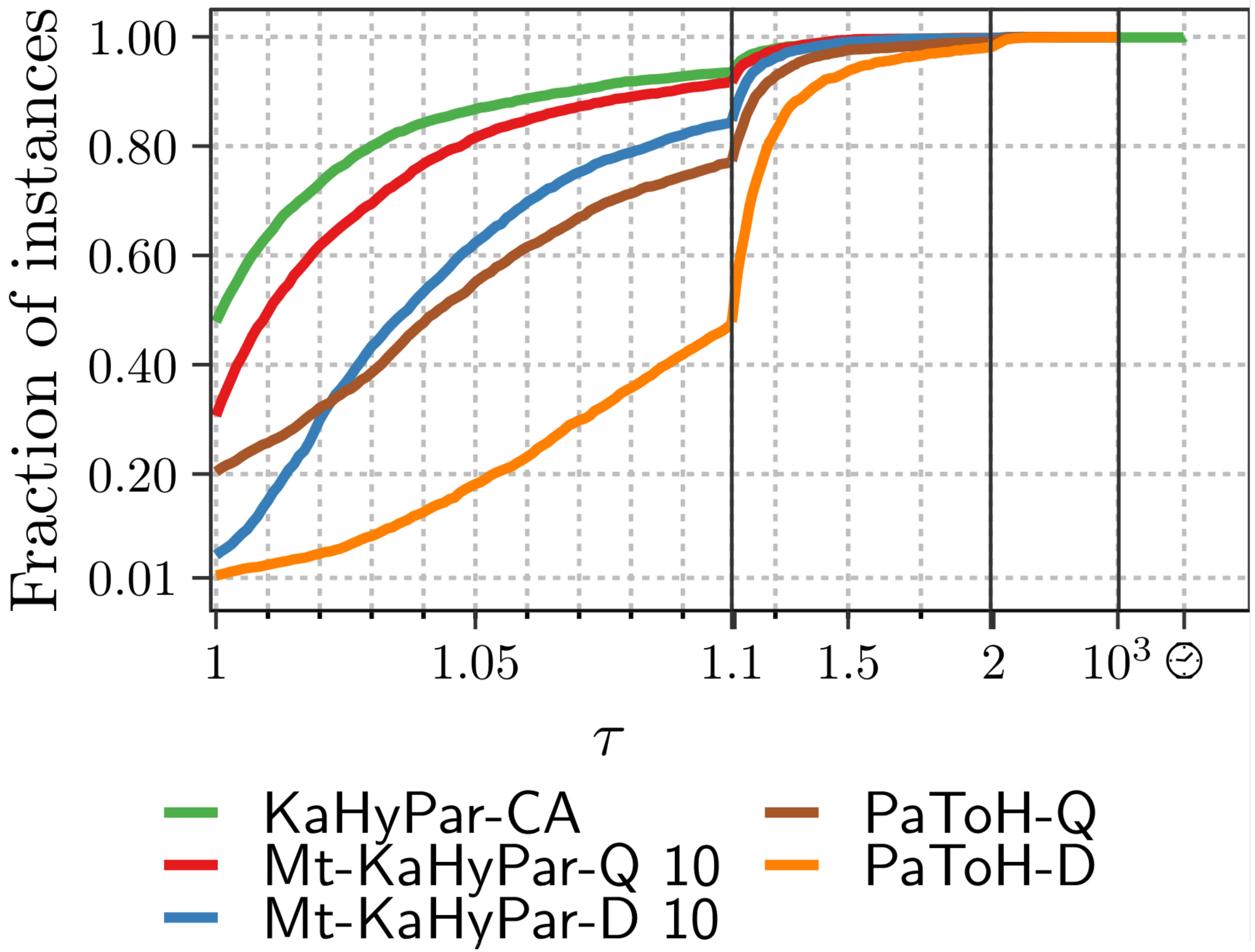
- $k \in \{2, 4, 8, 16, 32, 64, 128\}$ with imbalance: $\varepsilon = 3\%$
- 10 random seeds
- 10 threads

Experiments – Connectivity Metric (Quality)



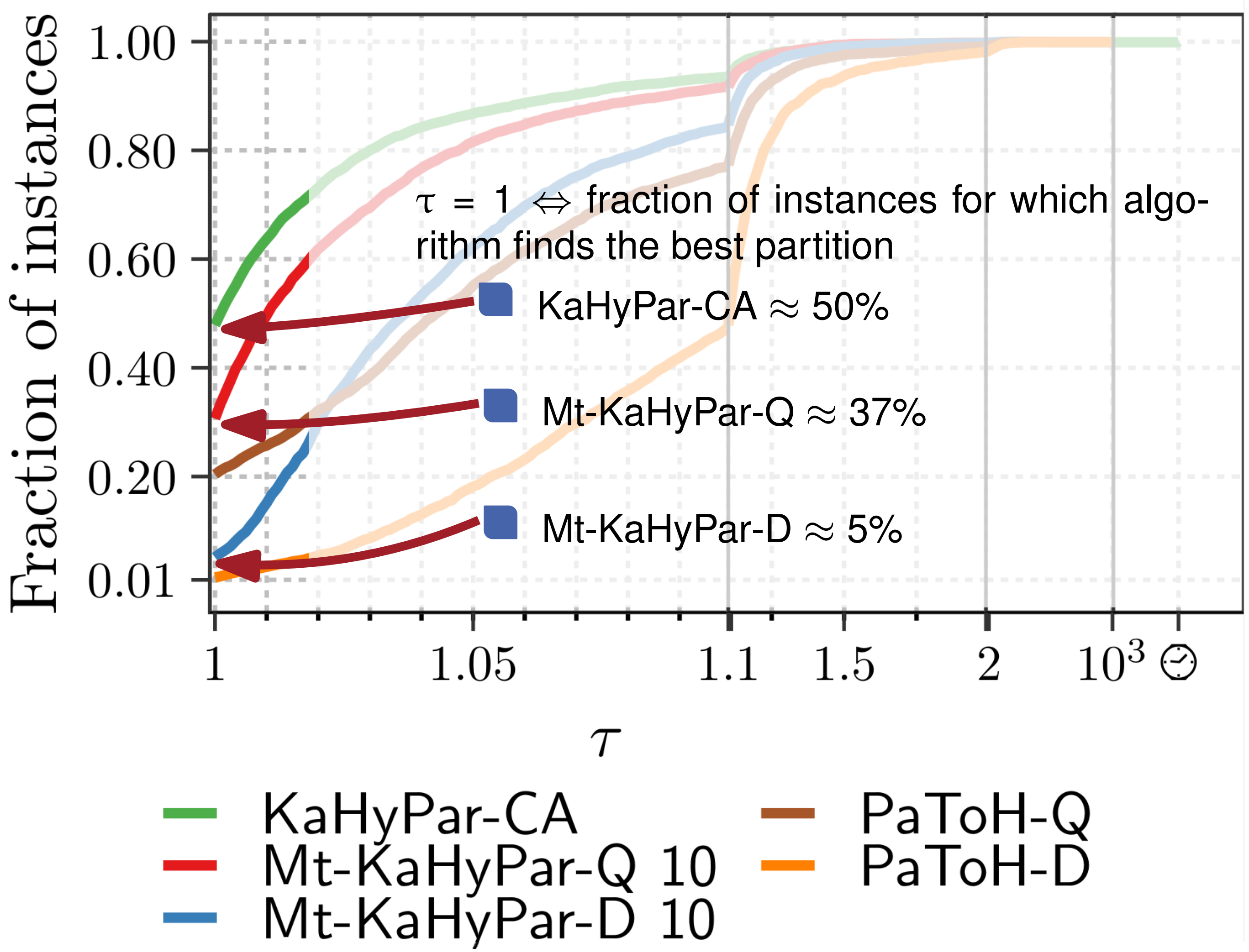
Experiments – Connectivity Metric (Quality)

$$p_{Algo}(\tau) = |\{I \in \mathcal{I} \mid Algo(I) \leq \tau \cdot Best(I)\}| / |\mathcal{I}|$$



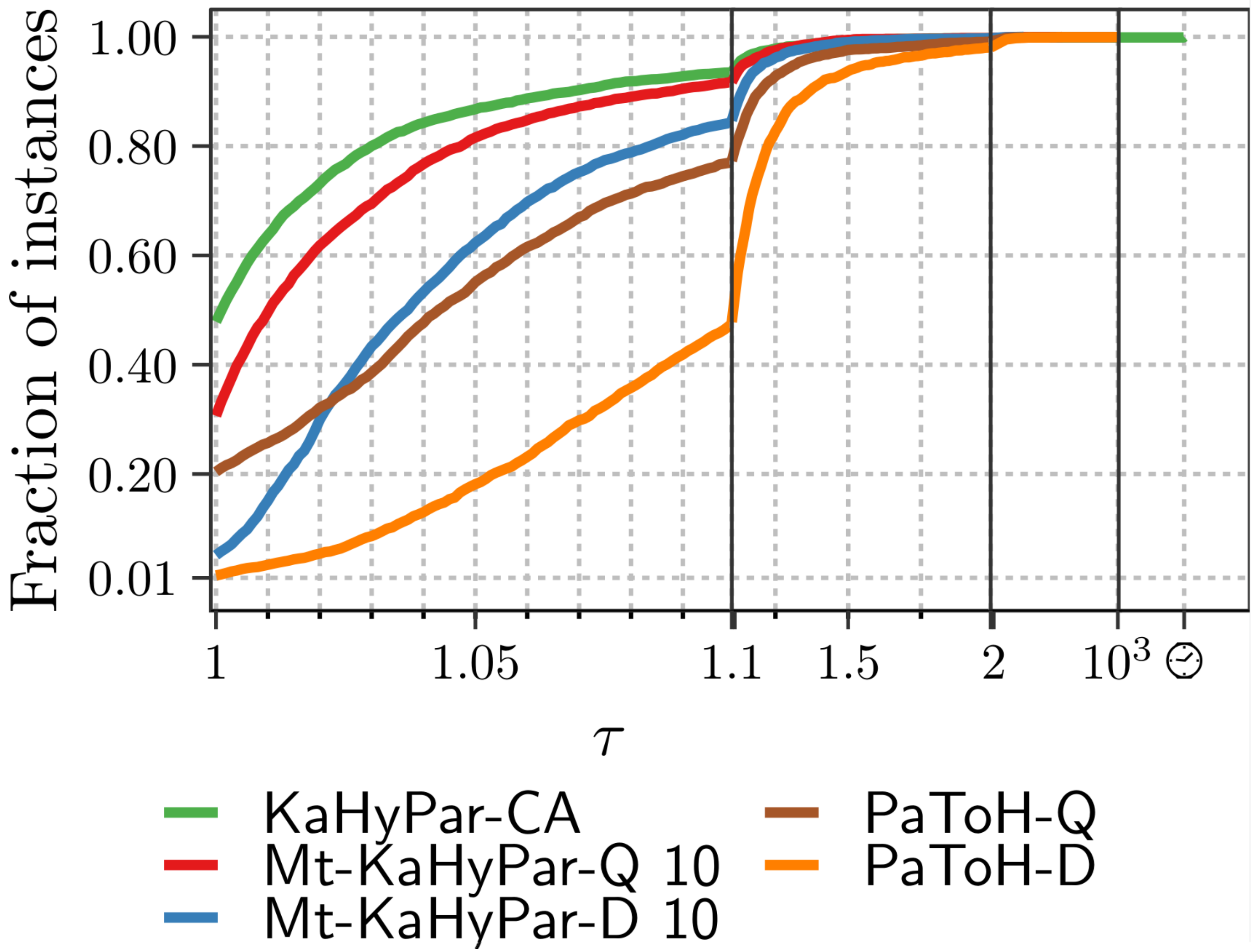
Experiments – Connectivity Metric (Quality)

$$p_{Algo}(\tau) = |\{I \in \mathcal{I} \mid Algo(I) \leq \tau \cdot Best(I)\}| / |\mathcal{I}|$$

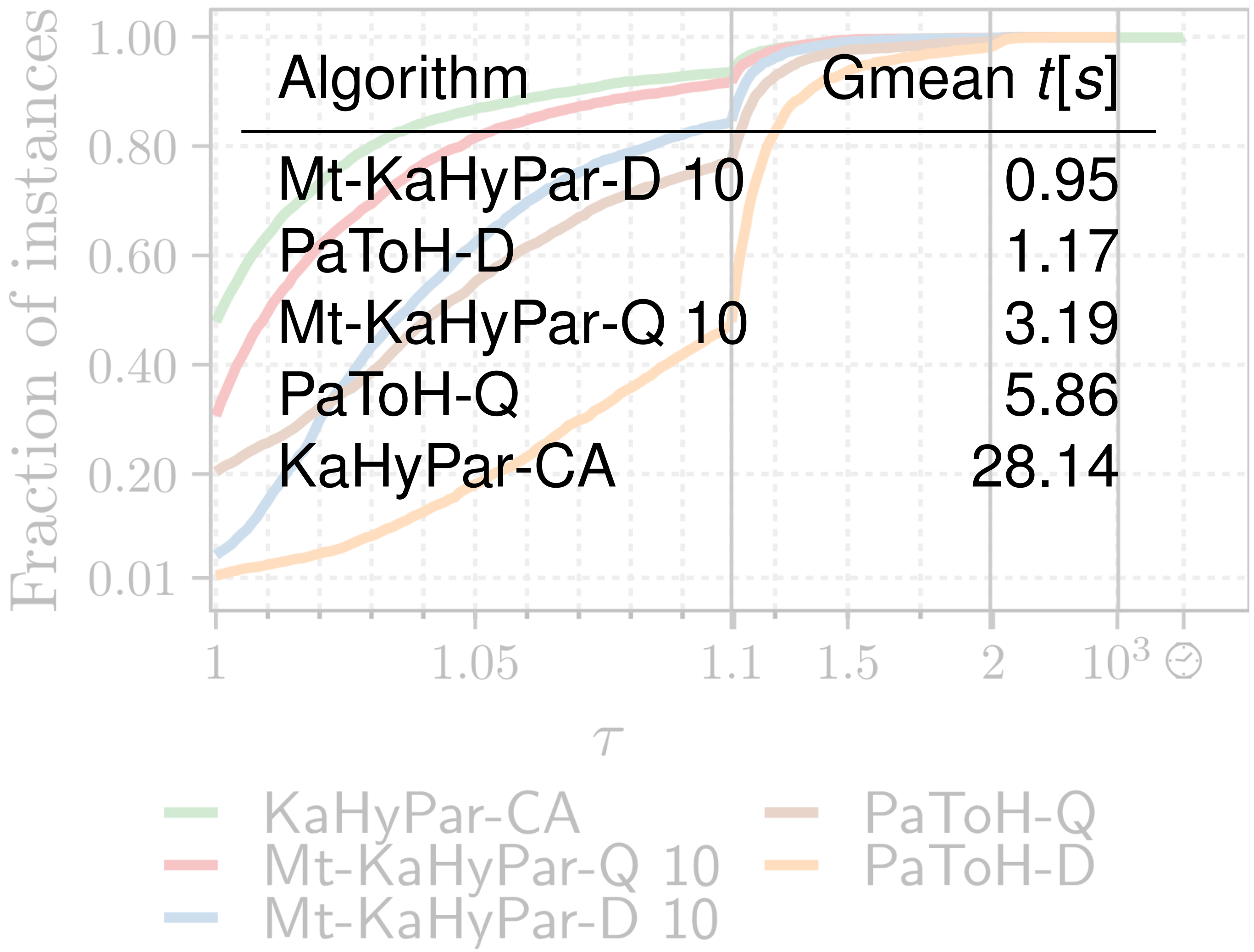


Experiments – Connectivity Metric (Quality)

$$p_{Algo}(\tau) = |\{I \in \mathcal{I} \mid Algo(I) \leq \tau \cdot Best(I)\}| / |\mathcal{I}|$$

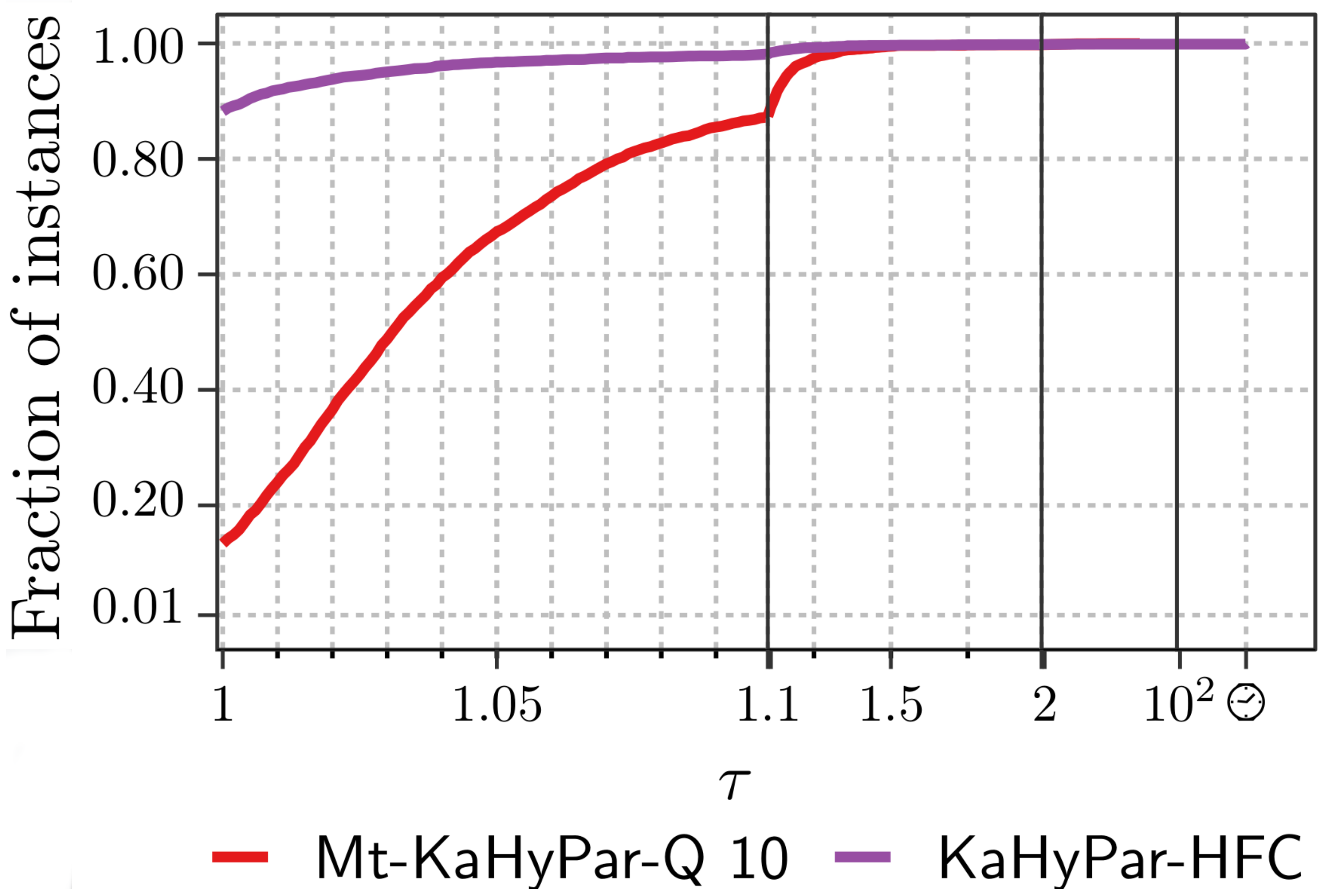


Experiments – Connectivity Metric (Quality)

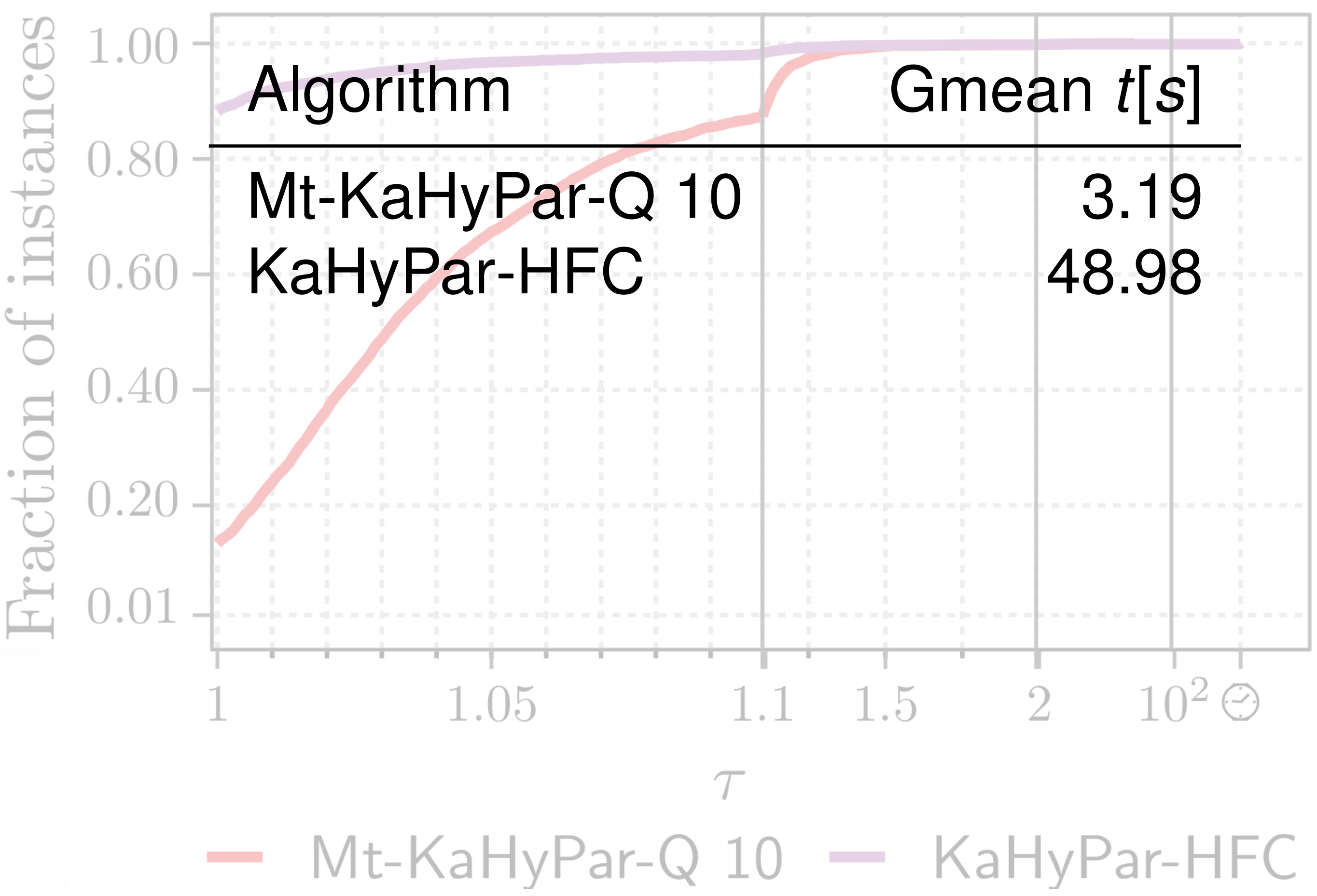


Algorithm	Gmean $t[s]$
Mt-KaHyPar-D 10	0.95
PaToH-D	1.17
Mt-KaHyPar-Q 10	3.19
PaToH-Q	5.86
KaHyPar-CA	28.14

$$p_{Algo}(\tau) = |\{I \in \mathcal{I} \mid Algo(I) \leq \tau \cdot Best(I)\}| / |\mathcal{I}|$$



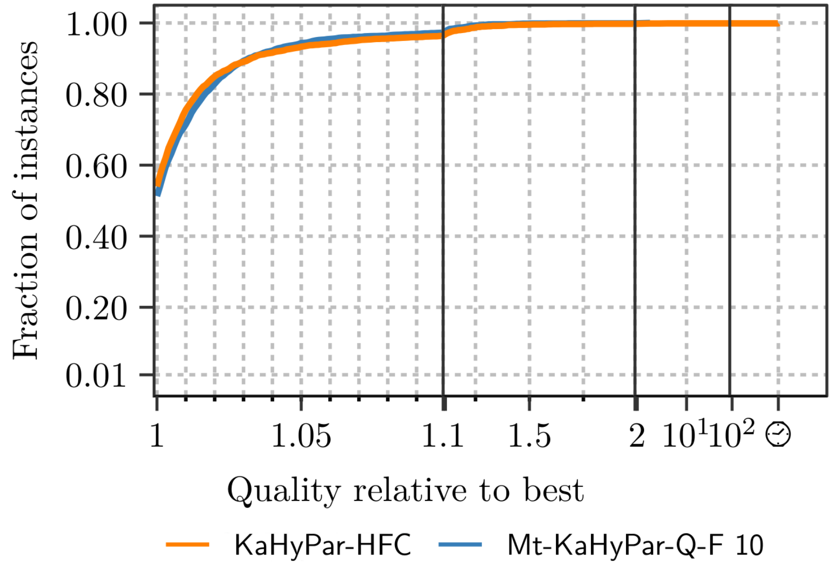
$p_{Algo}(\tau) = |\{I \in \mathcal{I} \mid Algo(I) \leq \tau \cdot Best(I)\}|/|\mathcal{I}|$



Algorithm	Gmean $t[s]$
Mt-KaHyPar-Q 10	3.19
KaHyPar-HFC	48.98

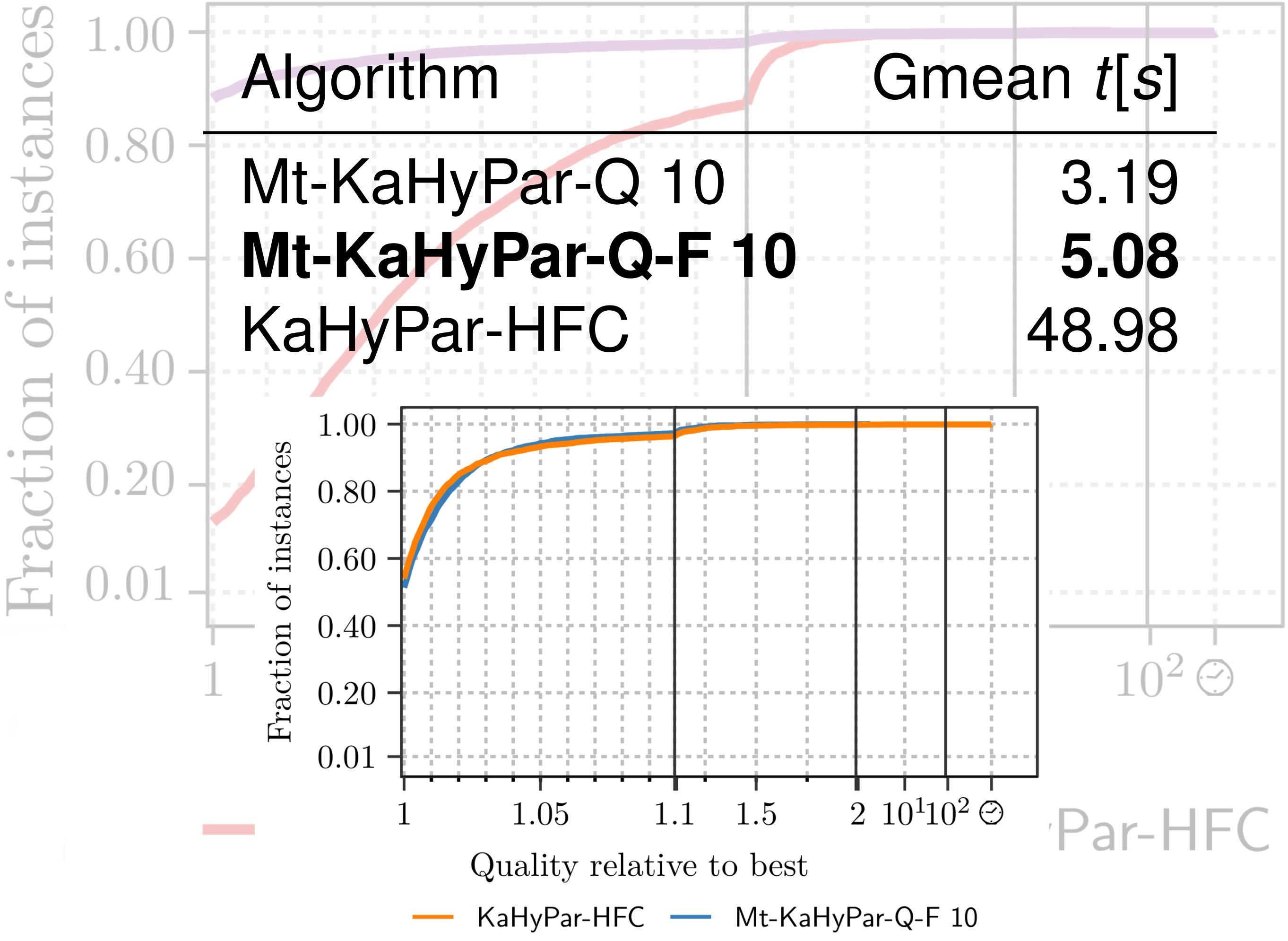
$$p_{Algo}(\tau) = |\{I \in \mathcal{I} \mid Algo(I) \leq \tau \cdot Best(I)\}| / |\mathcal{I}|$$

Algorithm	Gmean $t[s]$
Mt-KaHyPar-Q 10	3.19
Mt-KaHyPar-Q-F 10	**5.08**
KaHyPar-HFC	48.98

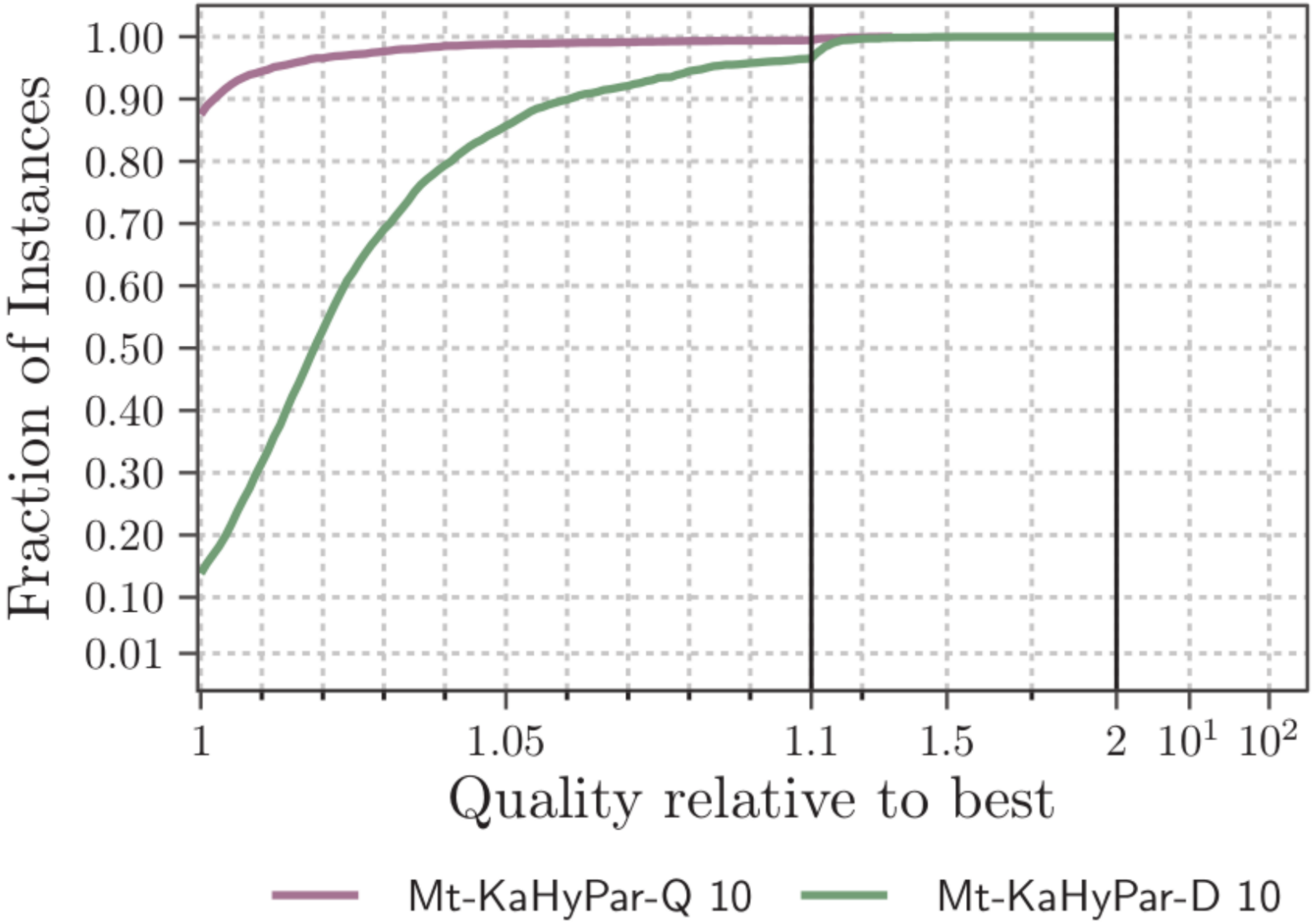


$$p_{Algo}(\tau) = |\{I \in \mathcal{I} \mid Algo(I) \leq \tau \cdot Best(I)\}|/|\mathcal{I}|$$

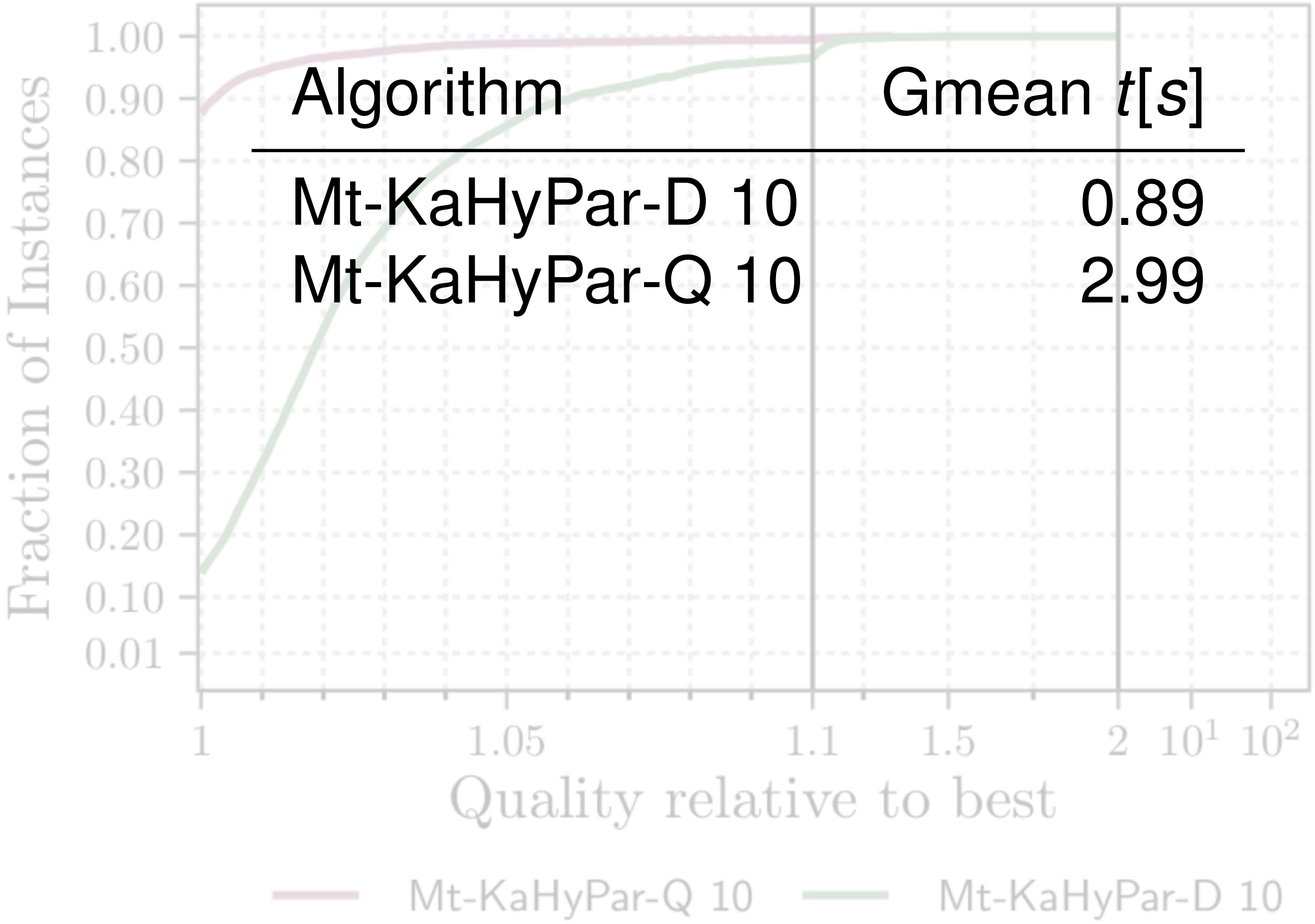
Algorithm	Gmean $t[s]$
Mt-KaHyPar-Q 10	3.19
Mt-KaHyPar-Q-F 10	**5.08**
KaHyPar-HFC	48.98



Multilevel vs *n*-Level Partitioning

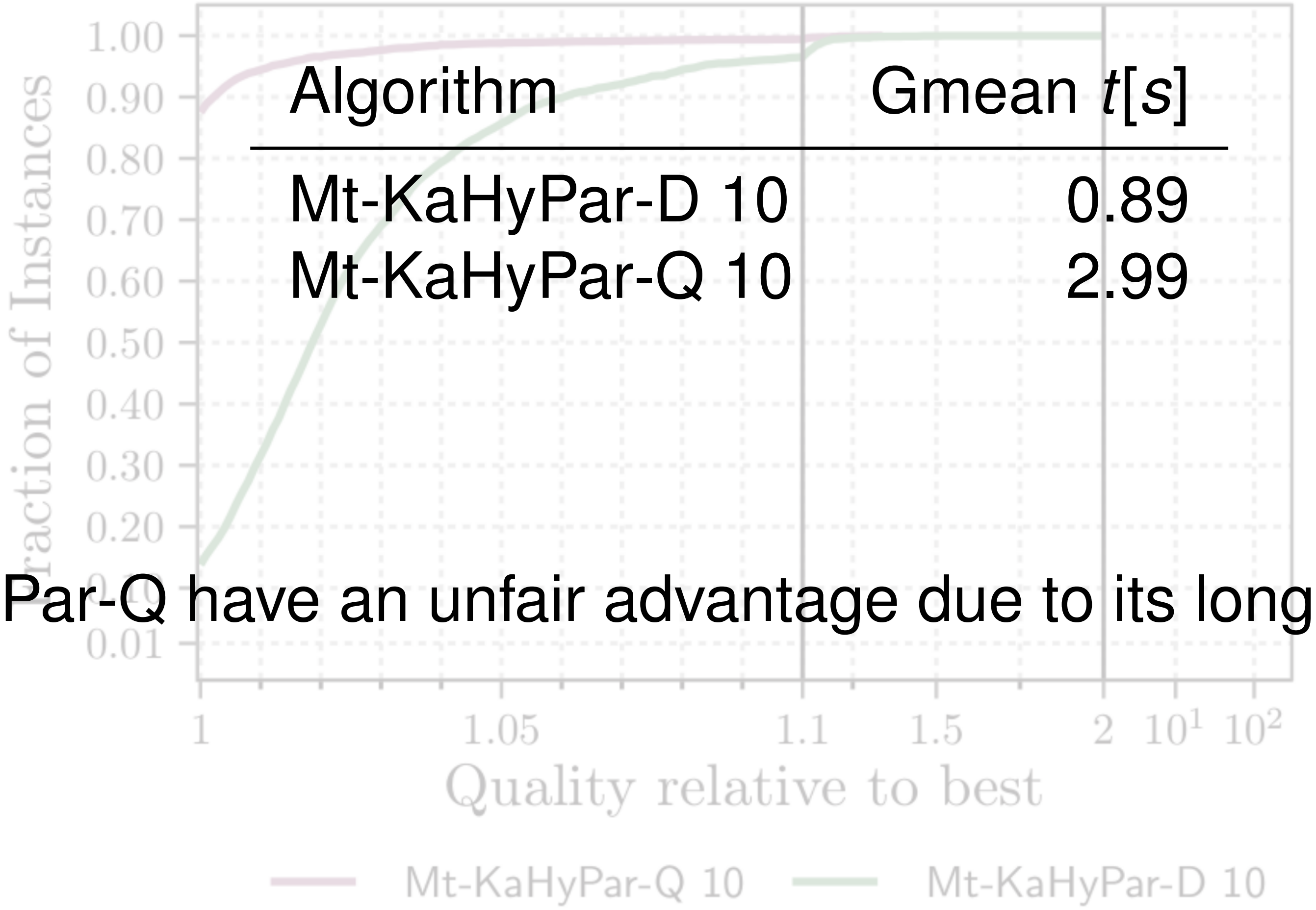


Multilevel vs *n*-Level Partitioning



Algorithm	Gmean $t[s]$
Mt-KaHyPar-D 10	0.89
Mt-KaHyPar-Q 10	2.99

Multilevel vs *n*-Level Partitioning



Algorithm	Gmean $t[s]$
Mt-KaHyPar-D 10	0.89
Mt-KaHyPar-Q 10	2.99

Does Mt-KaHyPar-Q have an unfair advantage due to its longer running time?

Effectiveness Tests

- **Idea**: Perform additional runs with the faster algorithm until its expected running time equals the running time of the slower algorithm

Effectiveness Tests

Karlsruhe Institute of Technology

- **Idea**: Perform additional runs with the faster algorithm until its expected running time equals the running time of the slower algorithm
- Given an instance I and two algorithms A and B

Effectiveness Tests

- **Idea**: Perform additional runs with the faster algorithm until its expected running time equals the running time of the slower algorithm
- Given an instance *I* and two algorithms *A* and *B*

Algorithm A

Run	1	2	3	4
Quality	1232	1123	1621	1345
Running Time	23.2	24.5	21.0	22.5

Algorithm B

Run	1	2	3	4
Quality	1532	1103	1287	1845
Running Time	5.2	8.3	6.0	7.3

Effectiveness Tests

- **Idea**: Perform additional runs with the faster algorithm until its expected running time equals the running time of the slower algorithm
- Given an instance *I* and two algorithms *A* and *B*

Sample one run from each algorithm

Algorithm A

Run	1	2	3	4
Quality	1232	1123	1621	1345
Running Time	23.2	24.5	21.0	22.5

Algorithm A

Best Result	1123
Total Time	24.5

Algorithm B

Run	1	2	3	4
Quality	1532	1103	1287	1845
Running Time	5.2	8.3	6.0	7.3

Algorithm B

Best Result	1845
Total Time	7.3

Effectiveness Tests

- **Idea**: Perform additional runs with the faster algorithm until its expected running time equals the running time of the slower algorithm
- Given an instance *I* and two algorithms *A* and *B*

Algorithm A

Run	1	2	3	4
Quality	1232	1123	1621	1345
Running Time	23.2	24.5	21.0	22.5

Algorithm B

Run	1	2	3	4
Quality	1532	1103	1287	1845
Running Time	5.2	8.3	6.0	7.3

Algorithm A

Best Result	1123
Total Time	24.5

Sample additional runs of algorithm B

Algorithm B

Best Result	1456
Total Time	11.6

Effectiveness Tests

Karlsruhe Institute of Technology

- **Idea**: Perform additional runs with the faster algorithm until its expected running time equals the running time of the slower algorithm
- Given an instance *I* and two algorithms *A* and *B*

Algorithm A

Run	1	2	3	4
Quality	1232	1123	1621	1345
Running Time	23.2	24.5	21.0	22.5

Algorithm B

Run	1	2	3	4
Quality	1532	1103	1287	1845
Running Time	5.2	8.3	6.0	7.3

Algorithm A

Best Result	1123
Total Time	24.5

Sample additional runs of algorithm B

Algorithm B

Best Result	1456
Total Time	16.8

Effectiveness Tests

Karlsruhe Institute of Technology

- **Idea**: Perform additional runs with the faster algorithm until its expected running time equals the running time of the slower algorithm
- Given an instance I and two algorithms A and B

Algorithm A

Run	1	2	3	4
Quality	1232	1123	1621	1345
Running Time	23.2	24.5	21.0	22.5

Algorithm A

Best Result	1123
Total Time	24.5

$16.8 + 8.3 = 25.1 > 24.5$

$\Rightarrow$ accept last sample with probability $(24.5 - 16.8)/8.3 = 92\%$

Algorithm B

Run	1	2	3	4
Quality	1532	1103	1287	1845
Running Time	5.2	8.3	6.0	7.3

Algorithm B

Best Result	1456
Total Time	16.8

Effectiveness Tests

Karlsruhe Institute of Technology

- **Idea**: Perform additional runs with the faster algorithm until its expected running time equals the running time of the slower algorithm
- Given an instance *I* and two algorithms *A* and *B*

Algorithm A

Run	1	2	3	4
Quality	1232	1123	1621	1345
Running Time	23.2	24.5	21.0	22.5

Algorithm B

Run	1	2	3	4
Quality	1532	1103	1287	1845
Running Time	5.2	8.3	6.0	7.3

Algorithm A

Best Result	1123
Total Time	24.5

Algorithm B

Best Result	1103
Total Time	25.1

Effectiveness Tests

Karlsruhe Institute of Technology

- **Idea**: Perform additional runs with the faster algorithm until its expected running time equals the running time of the slower algorithm
- Given an instance I and two algorithms A and B

Algorithm A

Run	1	2	3	4
Quality	1232	1123	1621	1345
Running Time	23.2	24.5	21.0	22.5

Algorithm B

Run	1	2	3	4
Quality	1532	1103	1287	1845
Running Time	5.2	8.3	6.0	7.3

This is also called a *virtual instance*
$\Rightarrow$ we create 10 virtual instances per instance

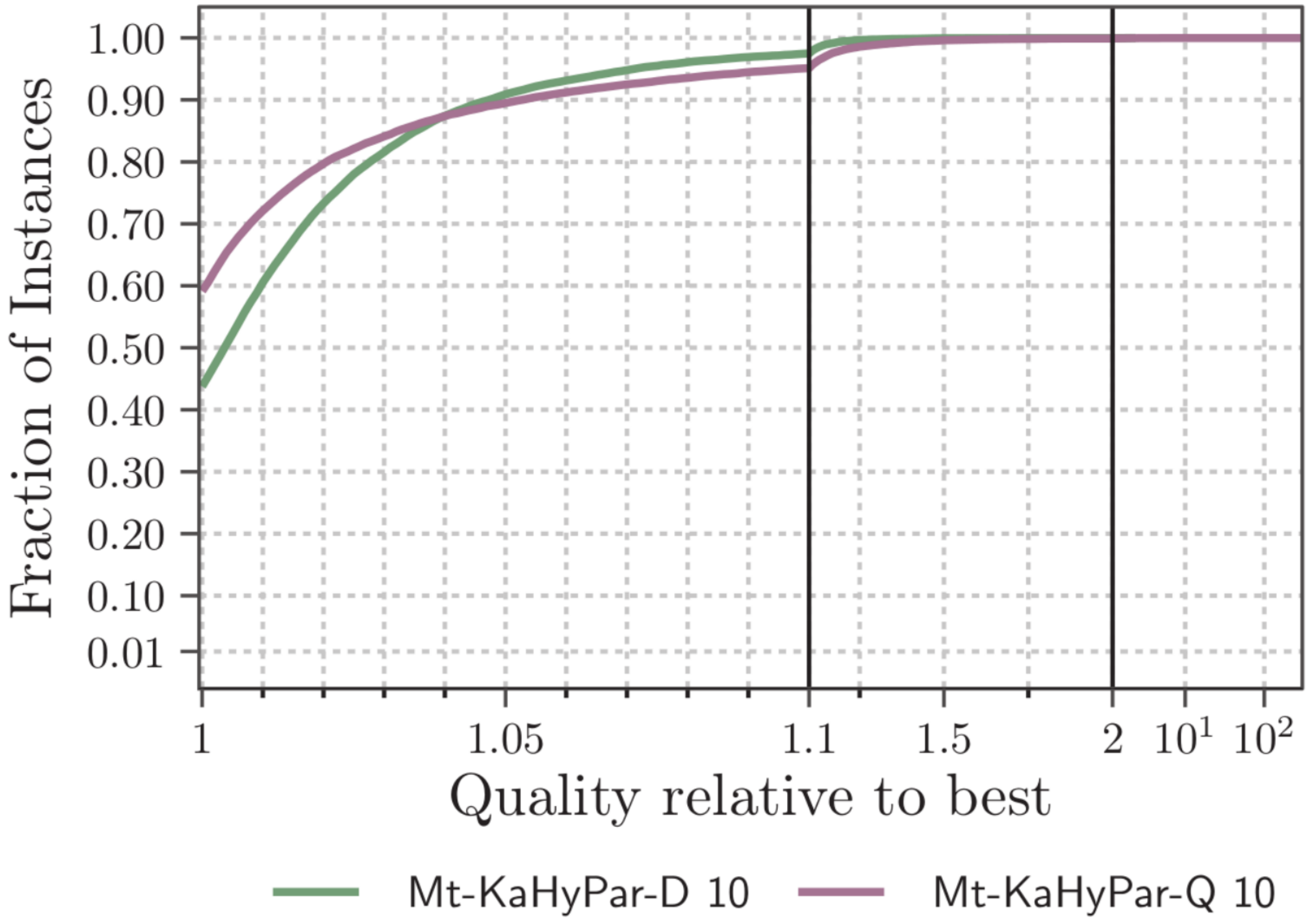
Algorithm A

Best Result	1123
Total Time	24.5

Algorithm B

Best Result	1103
Total Time	25.1

Multilevel vs *n*-Level - Effectiveness Tests



Conclusion

Mt-KaHyPar

`https://github.com/kahypar/mt-kahypar`

- achieves the **same solution quality** as the highest quality sequential system in fast parallel code
- **order of magnitude faster** than its sequential counterparts with only 10 threads
- great speedups