Karlsruhe Institute of Technology

Distributed Deep Multilevel Graph Partitioning

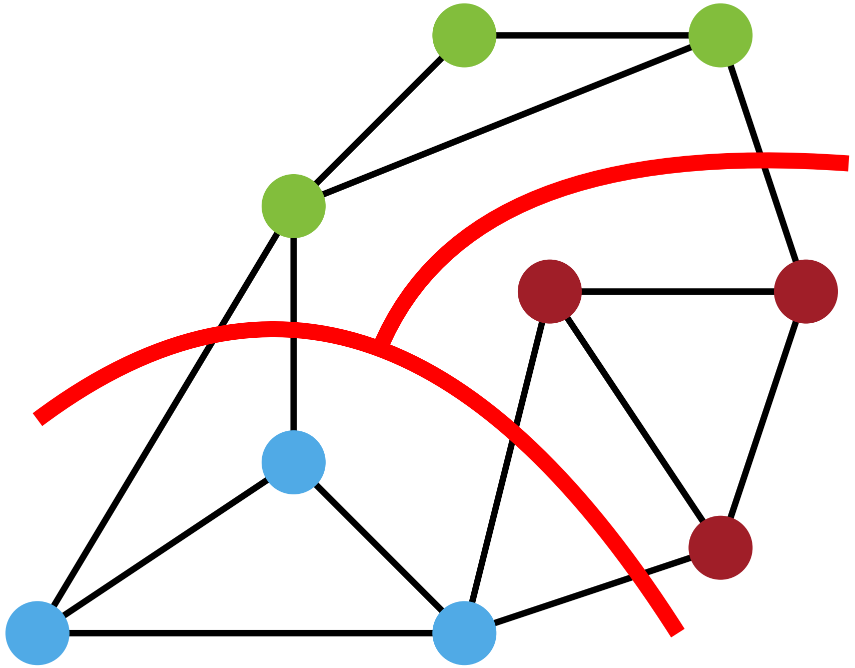
Euro-Par 2023 · August 31, 2023

Peter Sanders, Daniel Seemaier

Graph Partitioning

Given a graph $G = (V, E, c, \omega)$, partition V into k disjoint blocks such that:

- blocks have roughly the same weight: $c(V_i) \leq (1 + \varepsilon)\frac{c(V)}{k}$
- while minimizing the edge cut: $\sum_{i \neq j} \omega(E_{ij})$



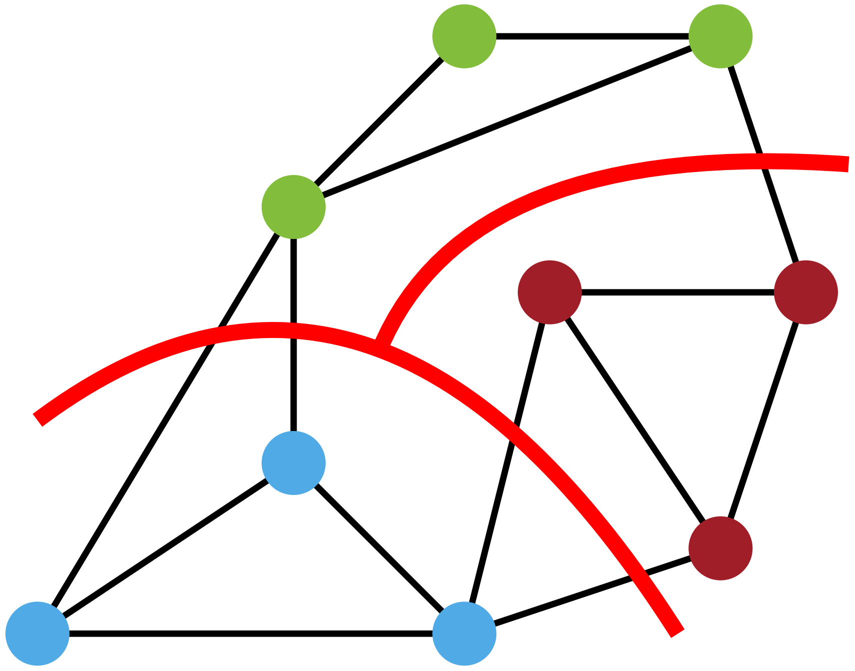
Graph Partitioning

Given a graph $G = (V, E, c, \omega)$, partition V into k disjoint blocks such that:

vertex weights

edge weights

- blocks have roughly the same weight: $c(V_i) \leq (1 + \varepsilon)\frac{c(V)}{k}$
- while minimizing the edge cut: $\sum_{i \neq j} \omega(E_{ij})$



Graph Partitioning

Given a graph $G = (V, E, c, \omega)$, partition V into k disjoint blocks such that:

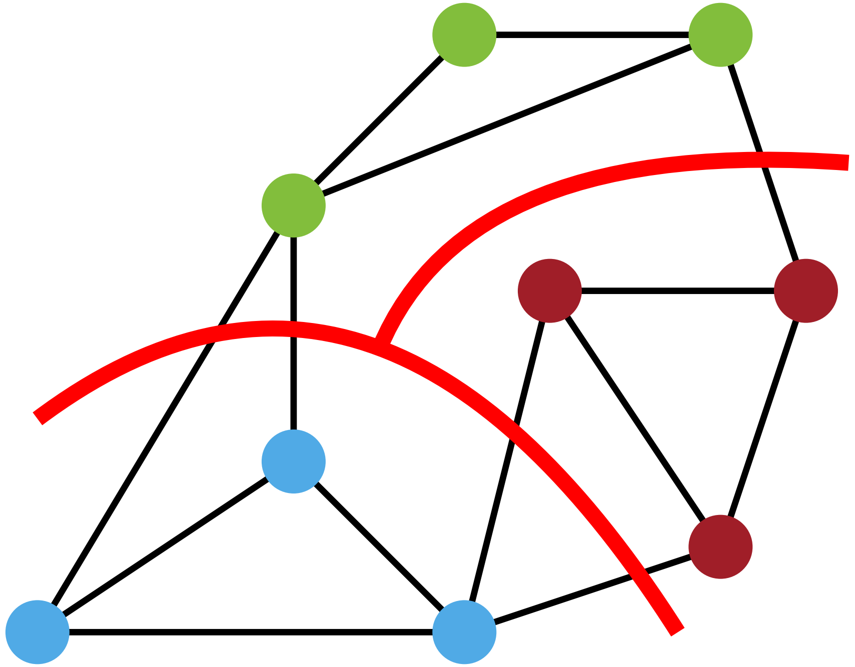
vertex weights

edge weights

- blocks have roughly the same weight: $c(V_i) \leq (1 + \varepsilon)\frac{c(V)}{k}$

imbalance factor

- while minimizing the edge cut: $\sum_{i \neq j} \omega(E_{ij})$



Graph Partitioning

Given a graph $G = (V, E, c, \omega)$, partition V into k disjoint blocks such that:

vertex weights

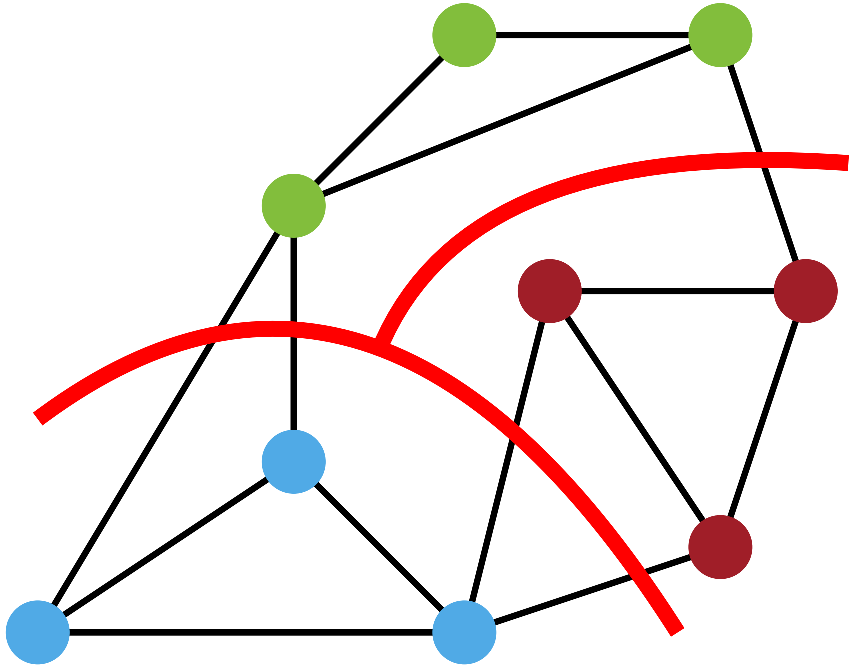
edge weights

- blocks have roughly the same weight: $c(V_i) \leq (1 + \varepsilon)\frac{c(V)}{k}$

imbalance factor

- while minimizing the edge cut: $\sum_{i \neq j} \omega(E_{ij})$

edges between blocks *i* and *j*



Graph Partitioning

Given a graph $G = (V, E, c, \omega)$, partition V into k disjoint blocks such that:

vertex weights

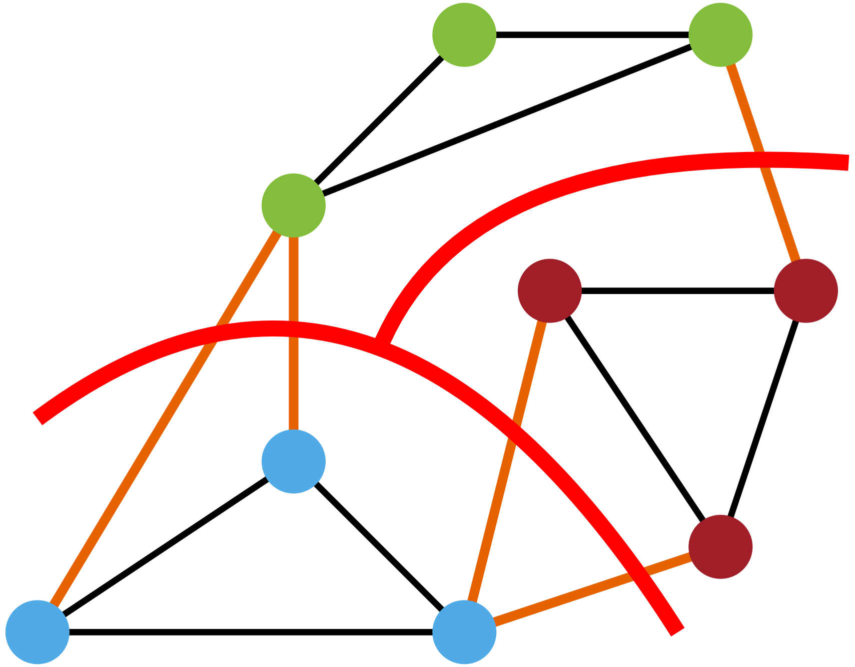
edge weights

- blocks have roughly the same weight: $c(V_i) \leq (1 + \varepsilon)\frac{c(V)}{k}$

imbalance factor

- while minimizing the edge cut: $\sum_{i \neq j} \omega(E_{ij})$ = 5

edges between blocks *i* and *j*



Graph Partitioning for Parallel Computing

Karlsruhe Institute of Technology

- Distributed graph processing:
  minimize communication between PEs

[HoreKa, KIT]

- Available parallelism increases steadily

- Established distributed GPs tools are not designed to handle large *k*

Graph Partitioning for Parallel Computing

- Distributed graph processing:
  minimize communication between PEs

[HoreKa, KIT]

- Available parallelism increases steadily

- Established distributed GPs tools are not designed to handle large *k*

contribution: improve scalability to large *k*

Graph Partitioning for Parallel Computing

- Distributed graph processing: minimize communication between PEs

[HoreKa, KIT]

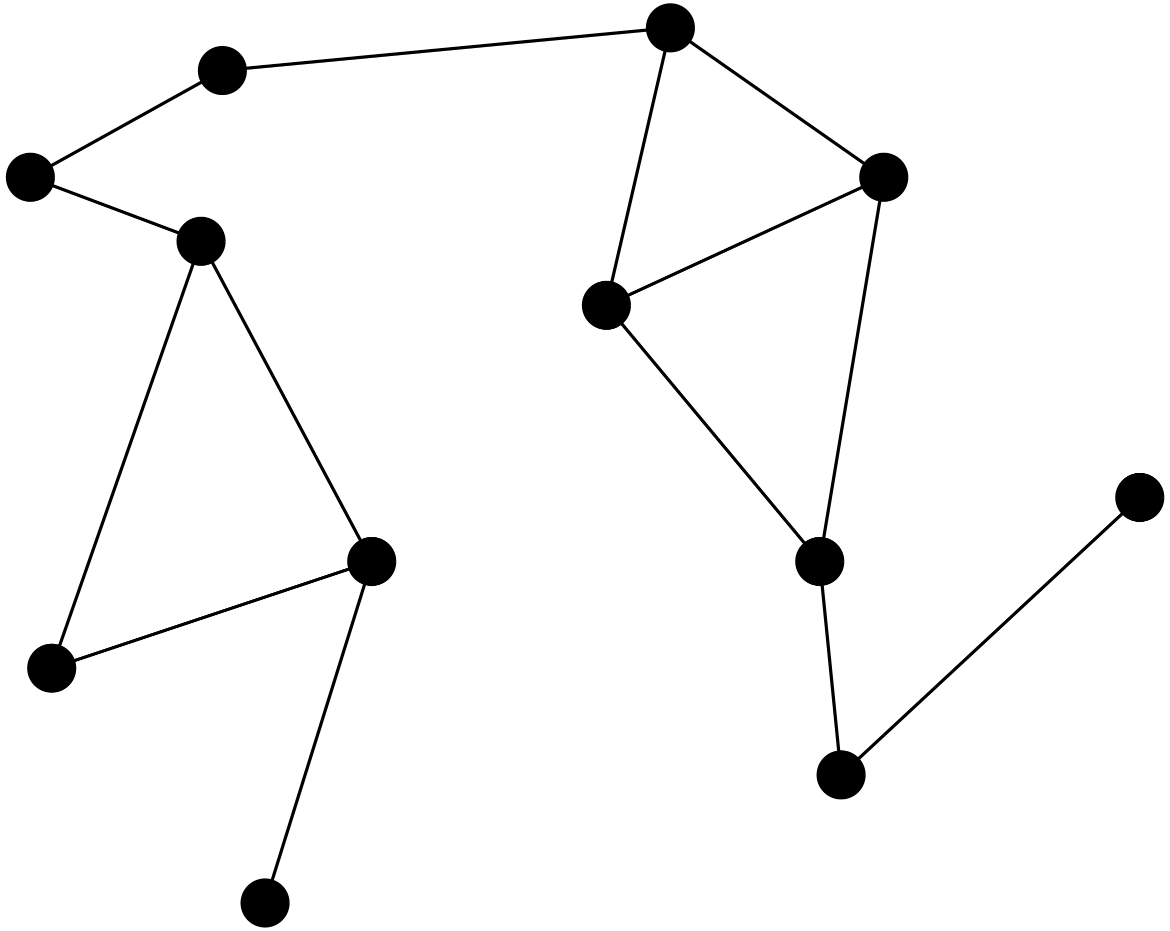
- Available parallelism increases steadily

Graph partitioning is NP-complete
⇒ we focus on heuristics

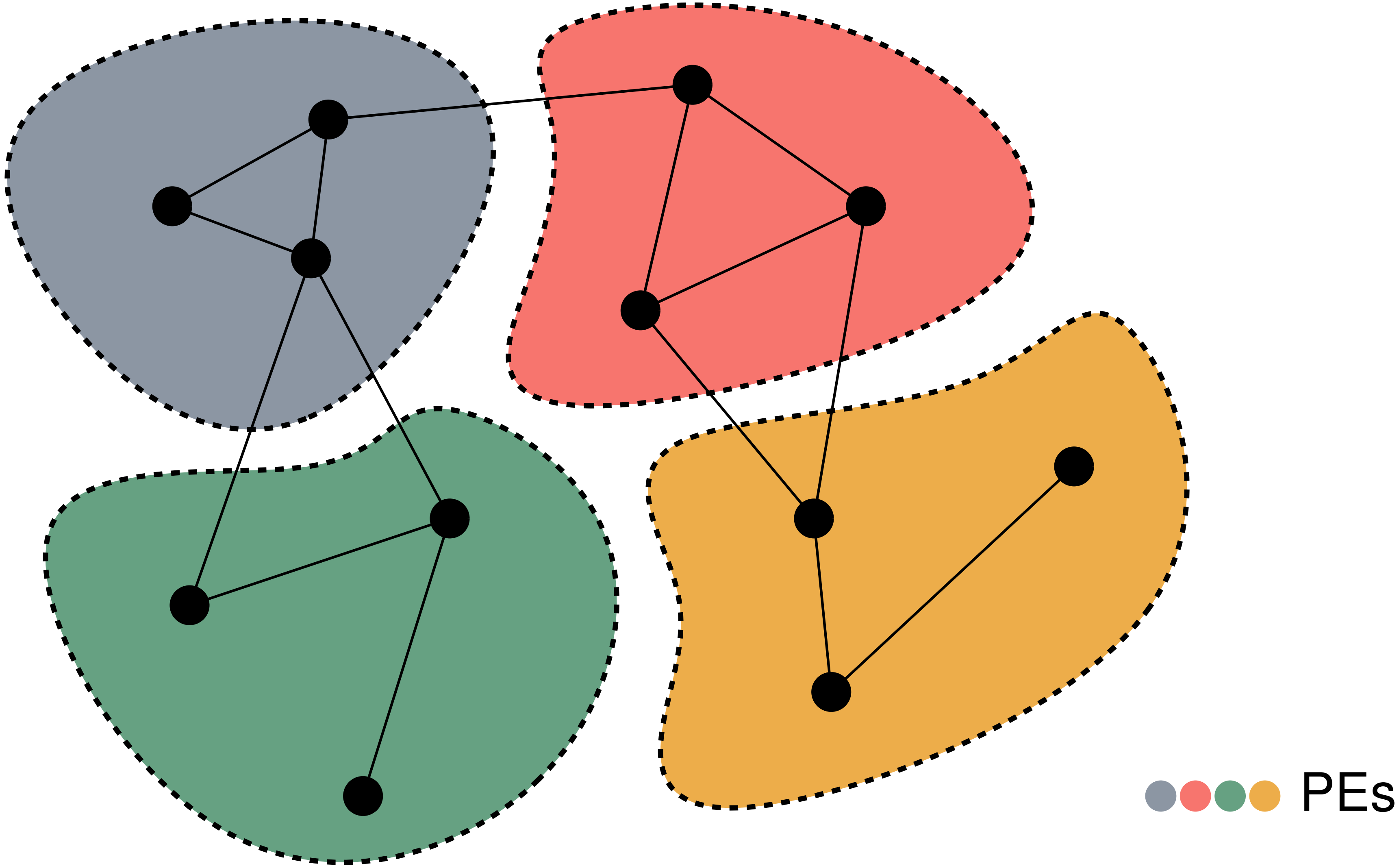
- Established distributed GPs tools are not designed to handle large *k*

contribution: improve scalability to large *k*

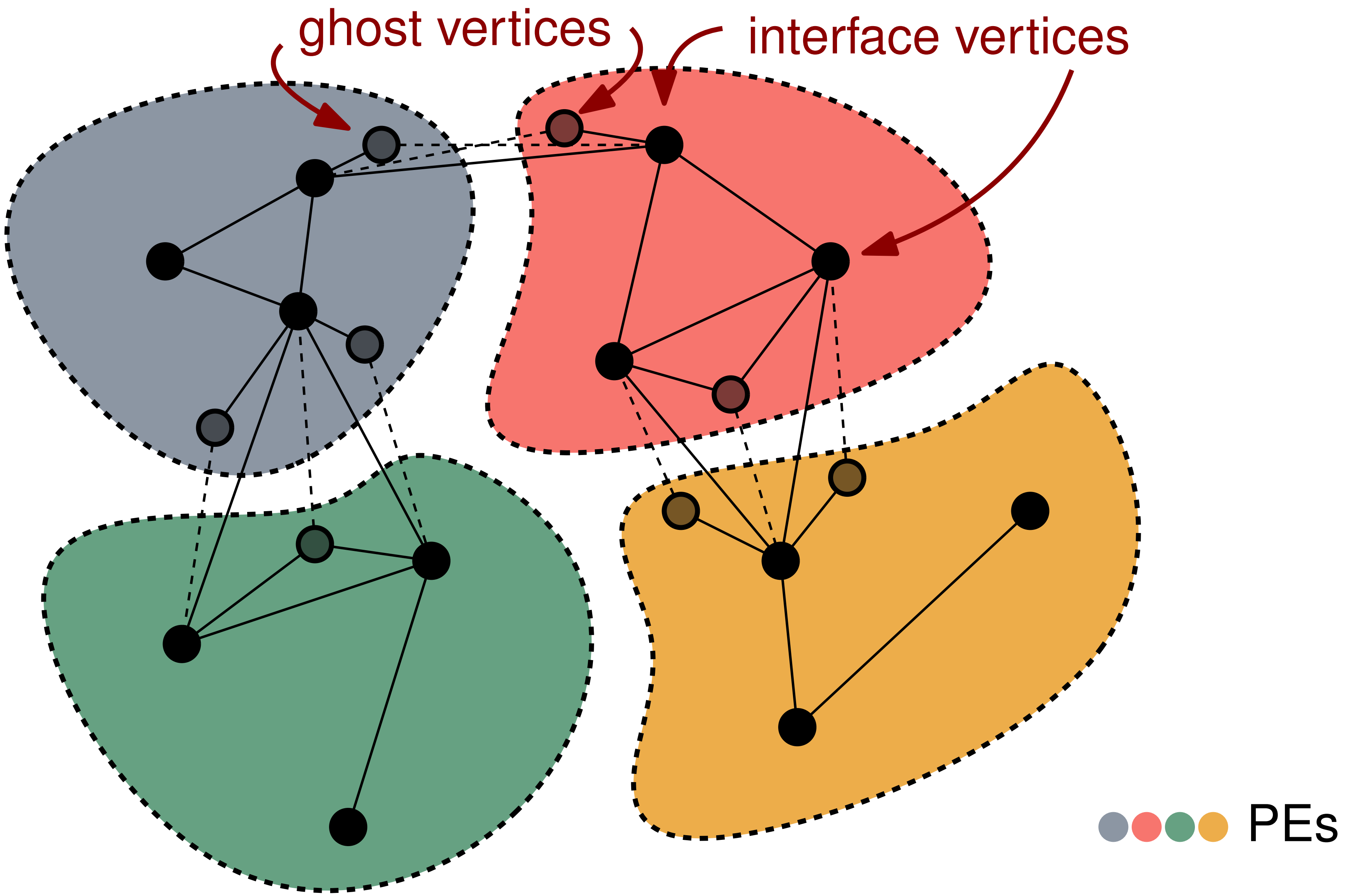
Graph Representation



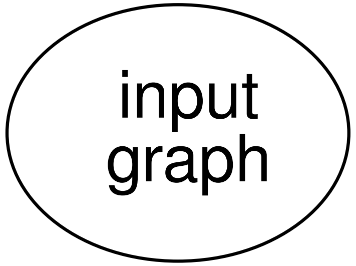
Graph Representation



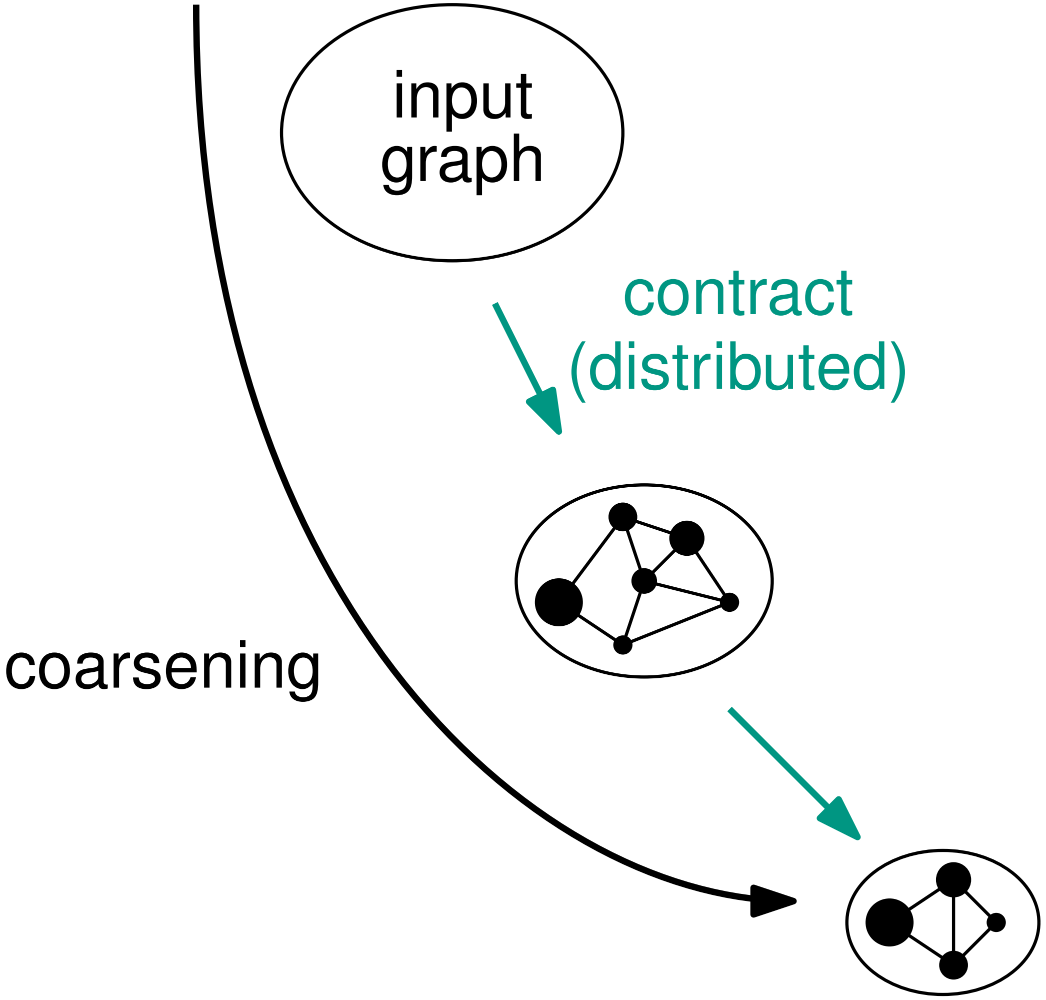
Graph Representation



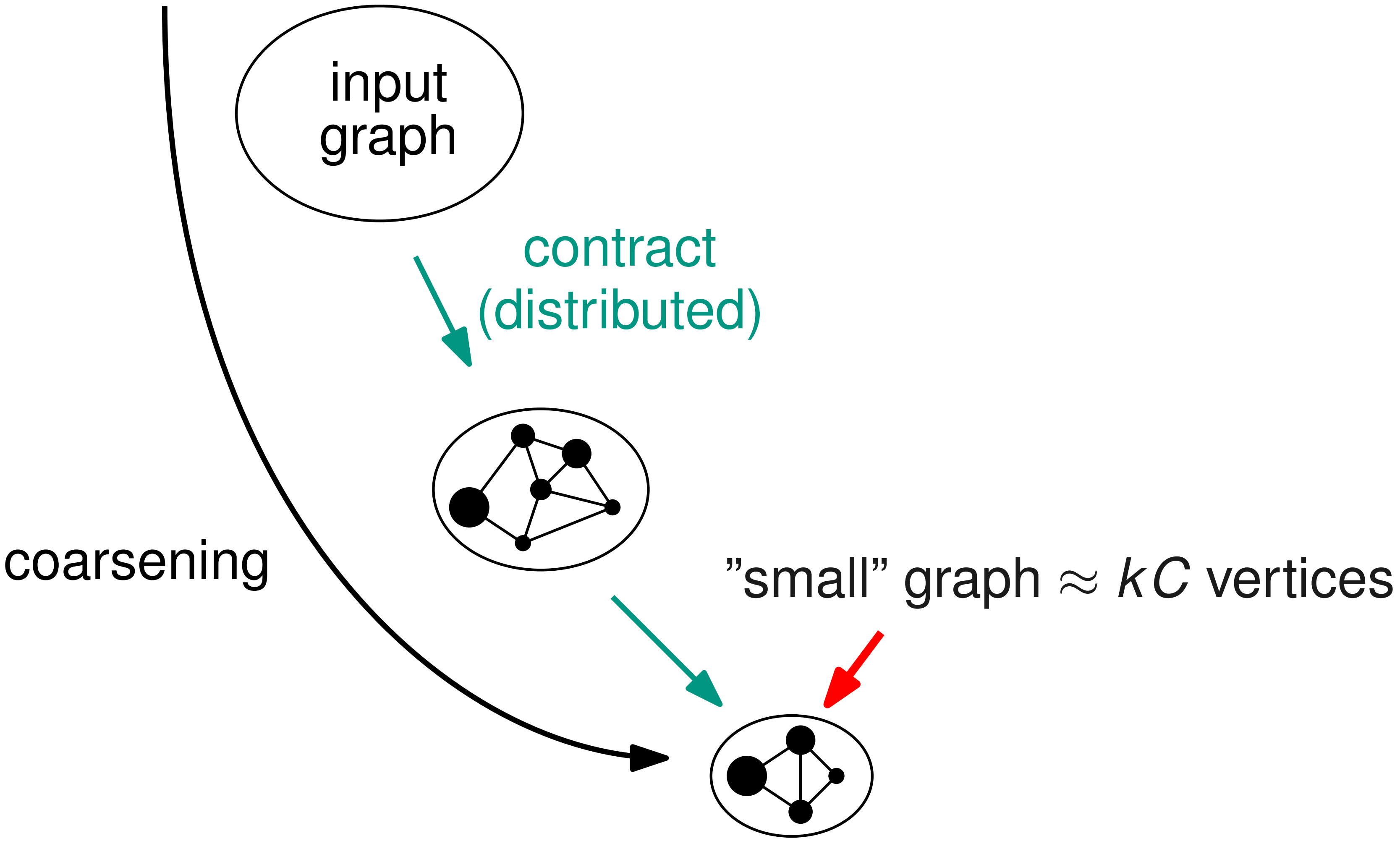
Multilevel Graph Partitioning



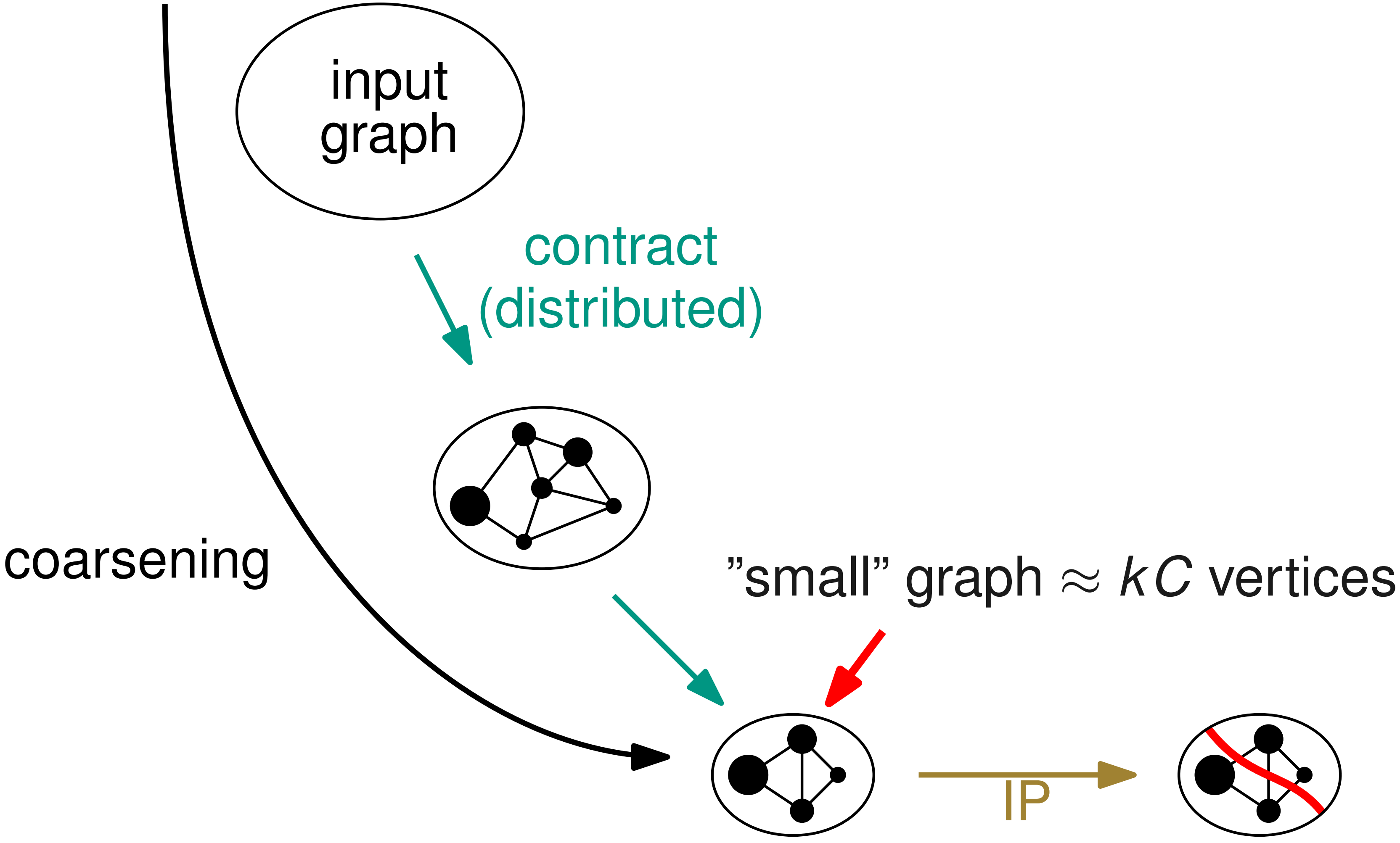
Multilevel Graph Partitioning



Multilevel Graph Partitioning



Multilevel Graph Partitioning



input graph

contract (distributed)

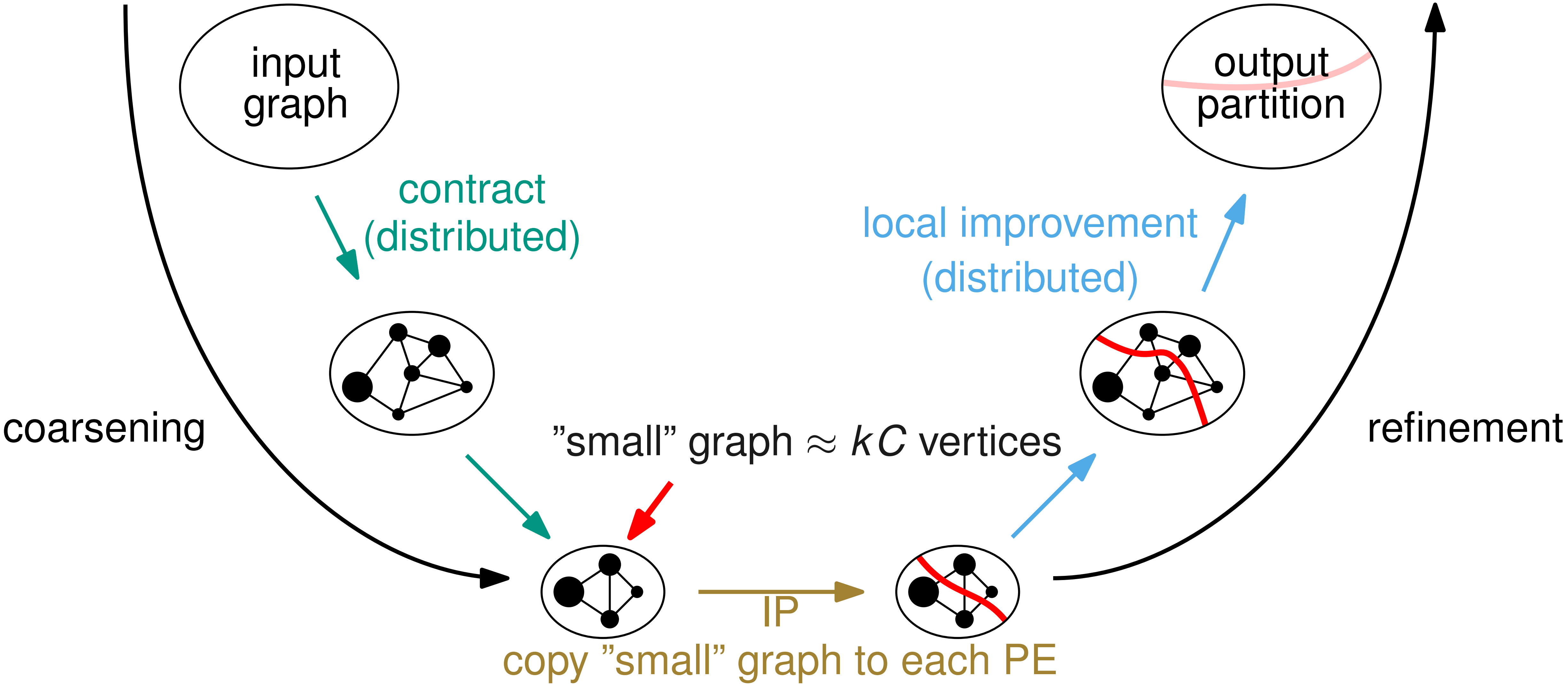
coarsening

"small" graph $\approx kC$ vertices

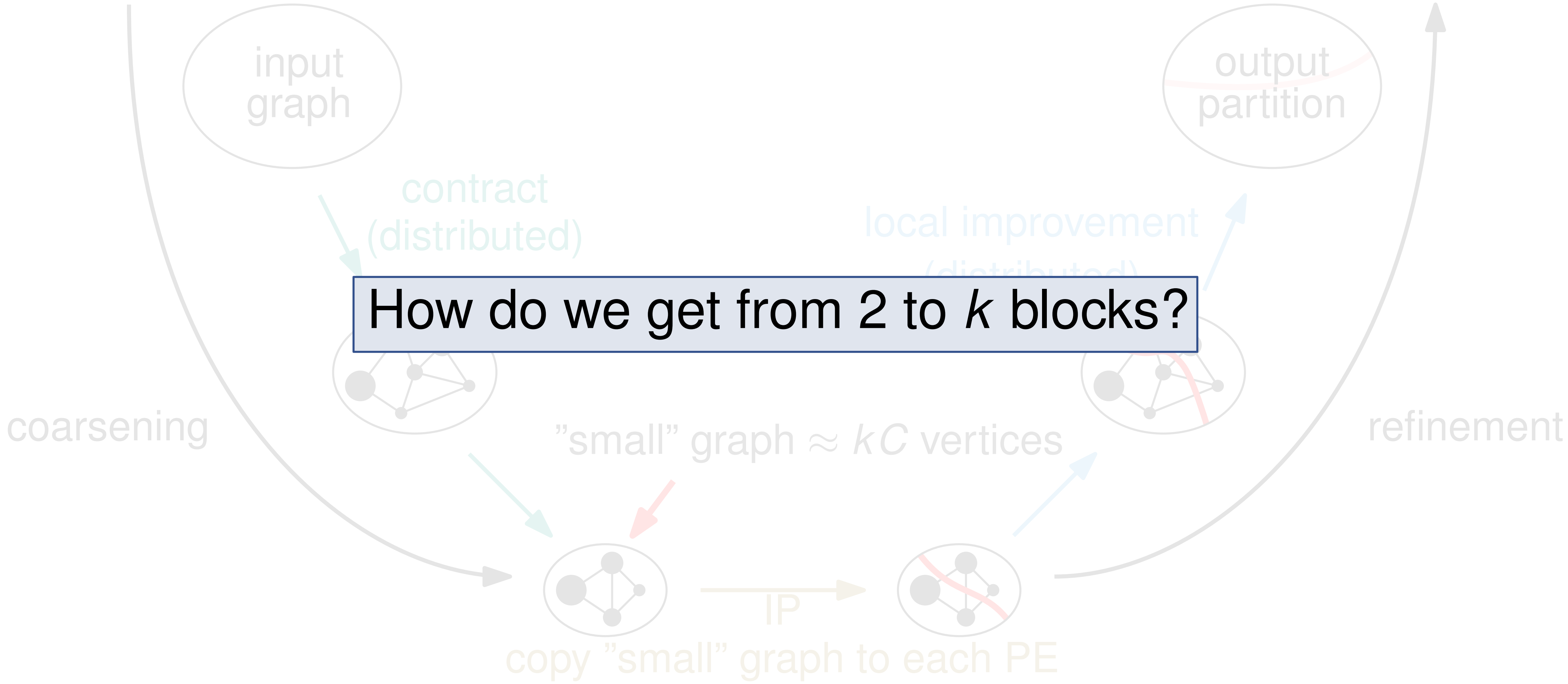
IP

copy "small" graph to each PE

Multilevel Graph Partitioning

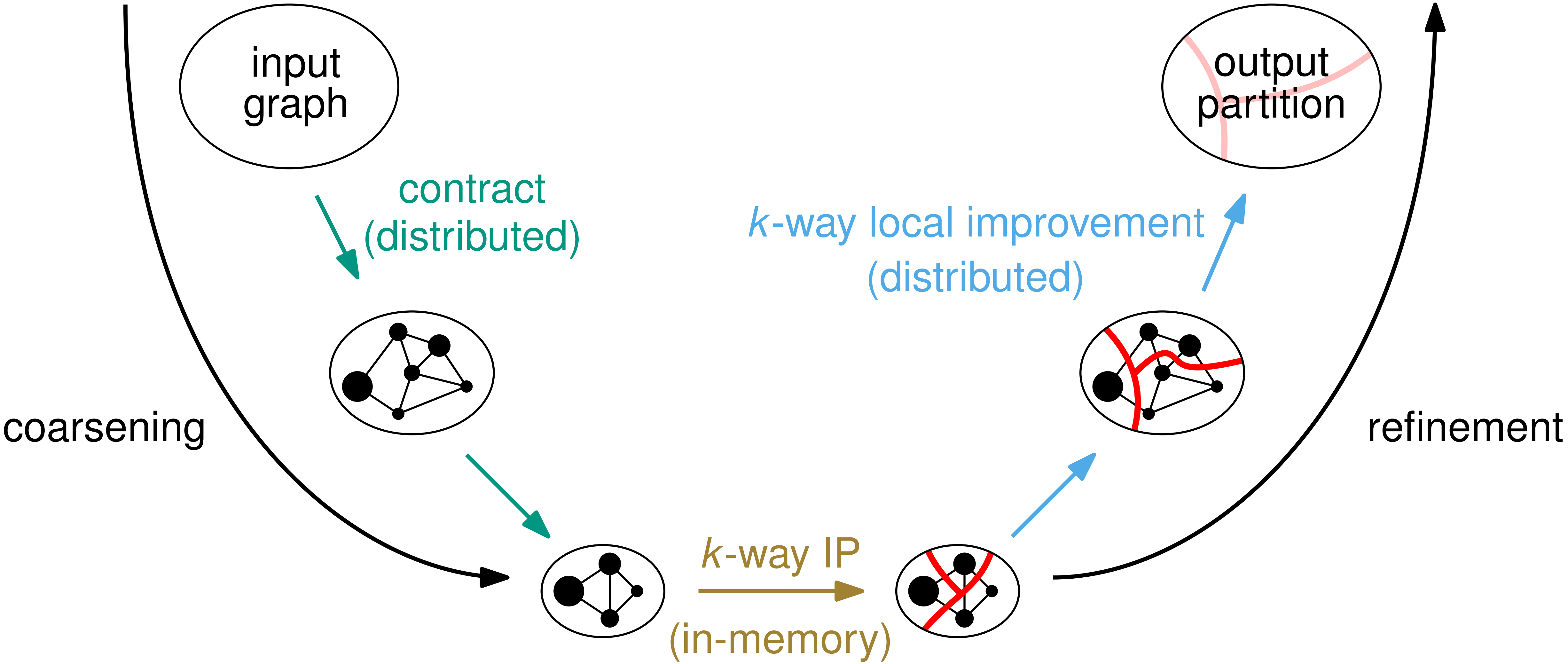

input graph
contract (distributed)
coarsening
"small" graph ≈ kC vertices
IP
copy "small" graph to each PE
local improvement (distributed)
output partition
refinement

Multilevel Graph Partitioning


input graph
output partition
contract (distributed)
local improvement (distributed)
coarsening
refinement
"small" graph ≈ kC vertices
IP
copy "small" graph to each PE

How do we get from 2 to k blocks?

MGP: Direct *k*-way



input graph

contract (distributed)

coarsening

k-way IP

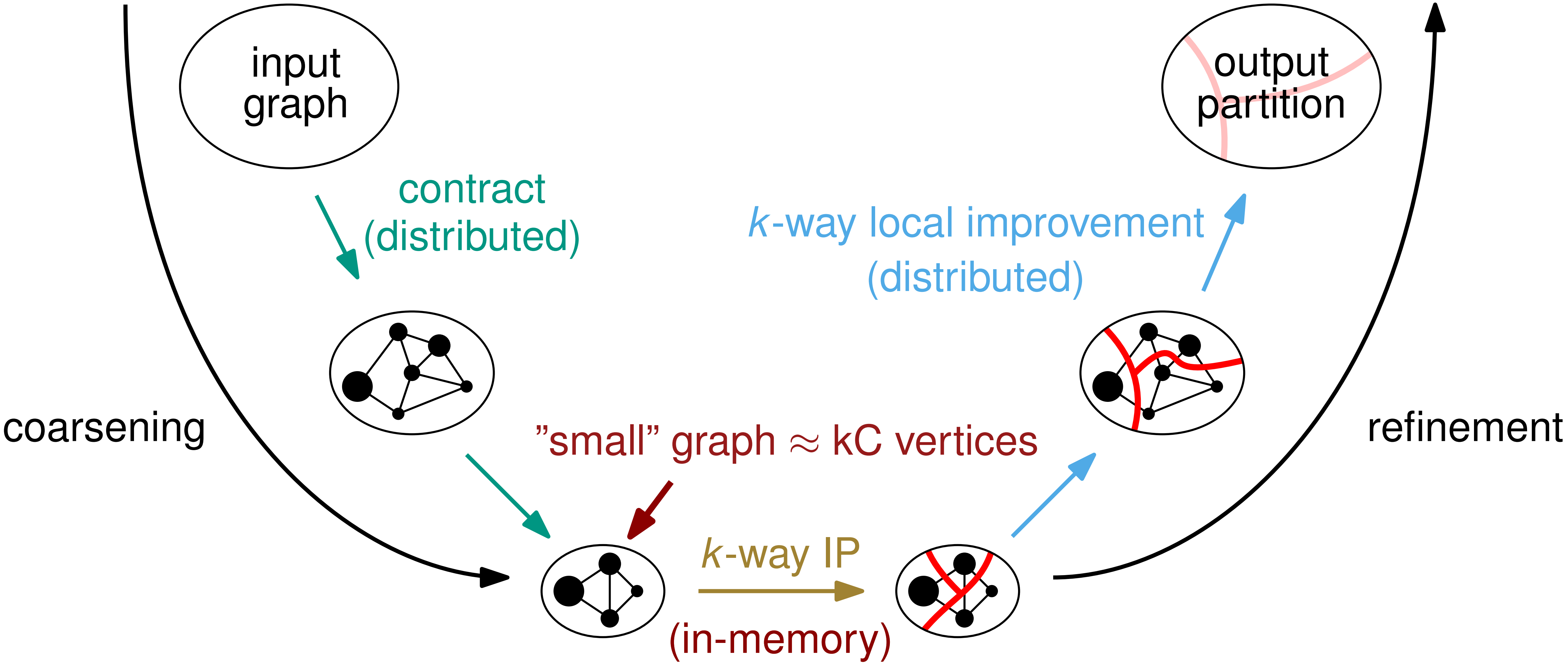
(in-memory)

k-way local improvement (distributed)

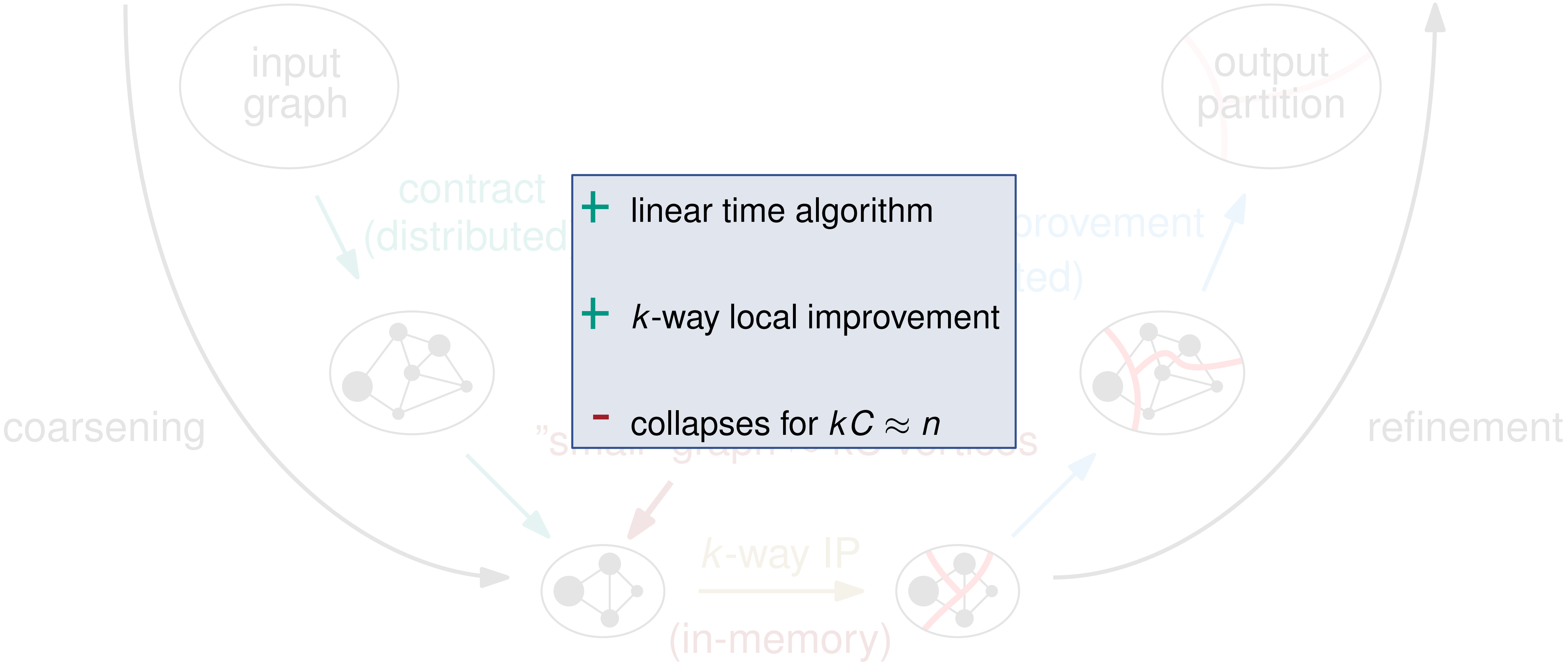
output partition

refinement

MGP: Direct *k*-way



MGP: Direct *k*-way

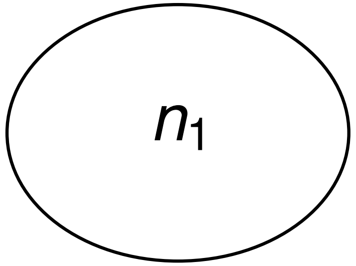


+ linear time algorithm

+ *k*-way local improvement

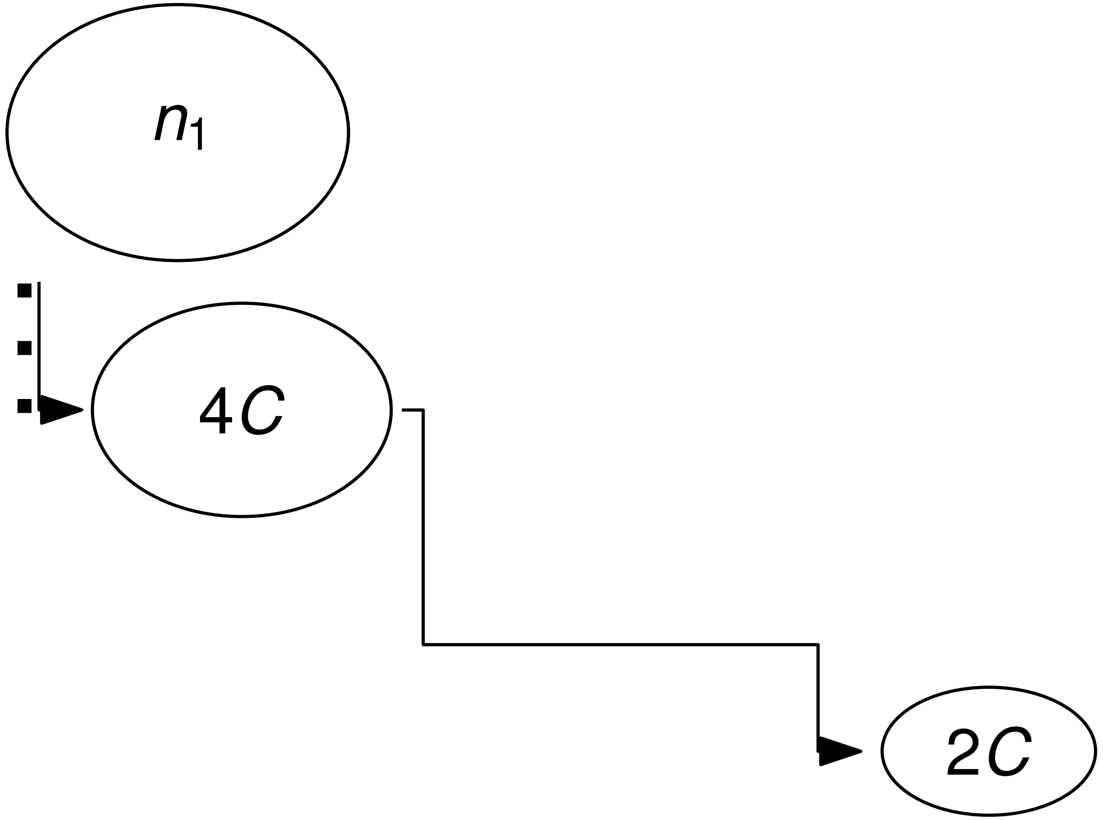
- collapses for $kC \approx n$

Distributed Deep MGP



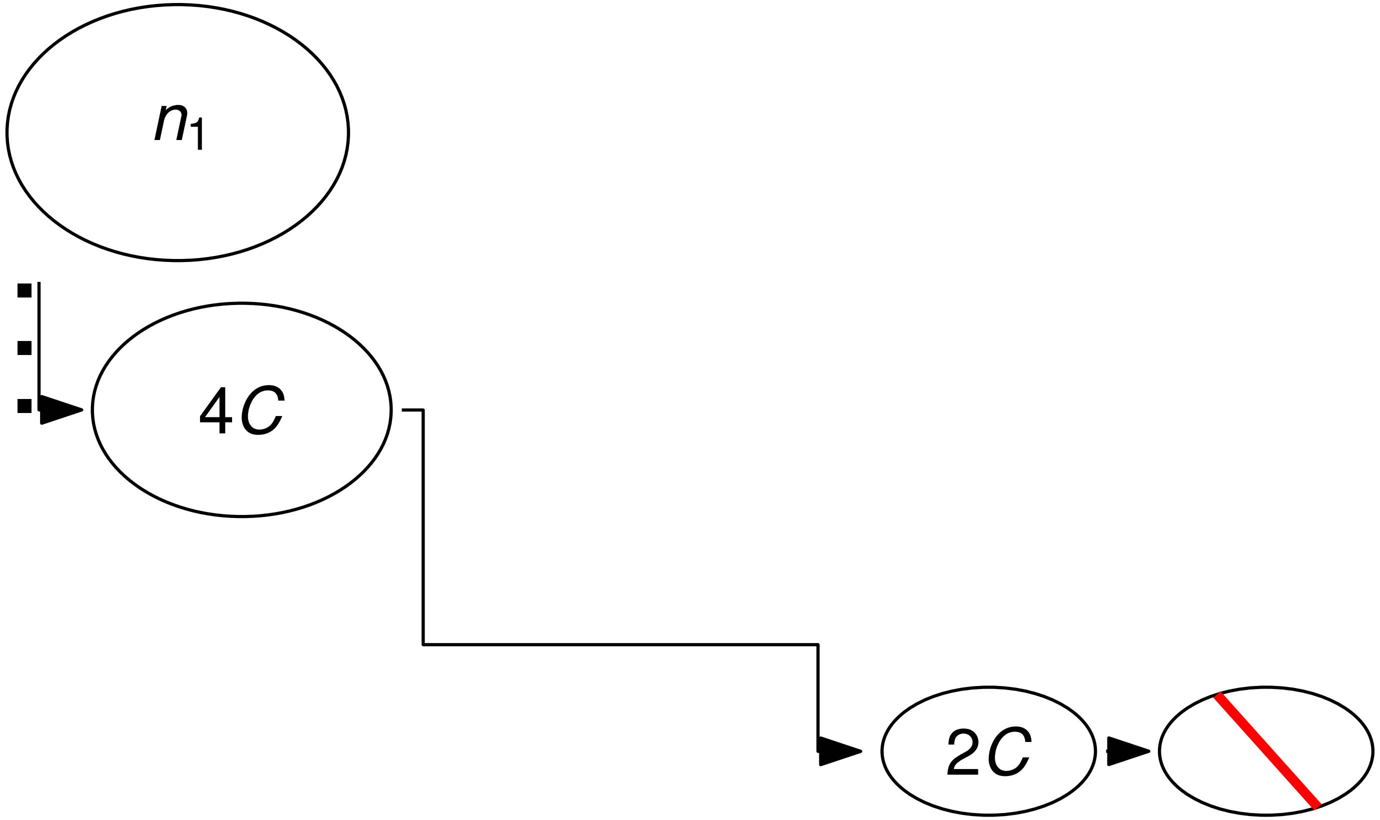
●●●● PEs

Distributed Deep MGP



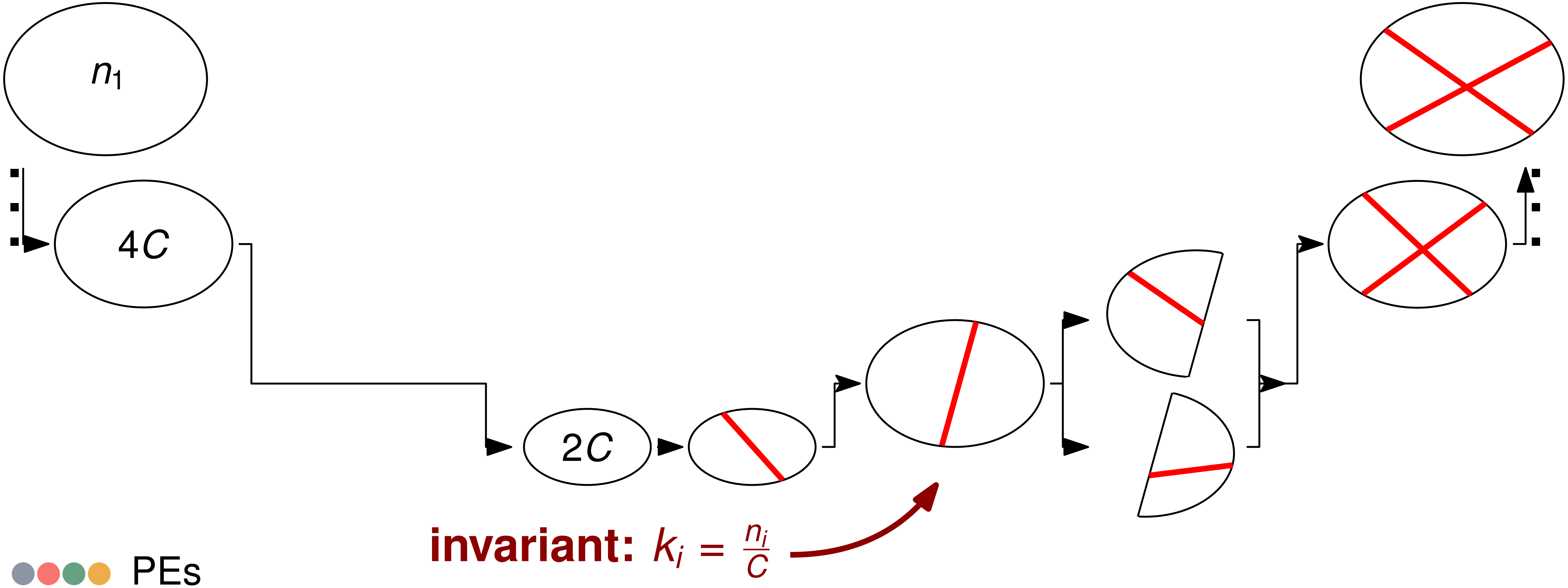
●●●● PEs

Distributed Deep MGP



●●●● PEs

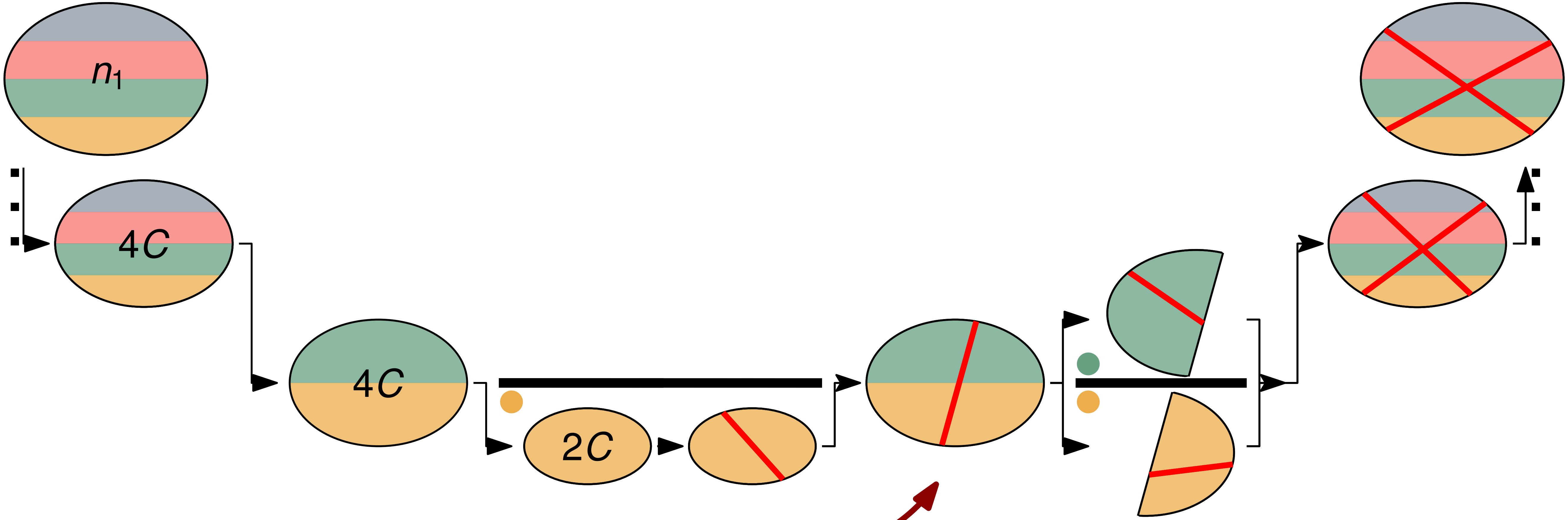
Distributed Deep MGP



invariant: $k_i = \frac{n_i}{C}$

●●●● PEs

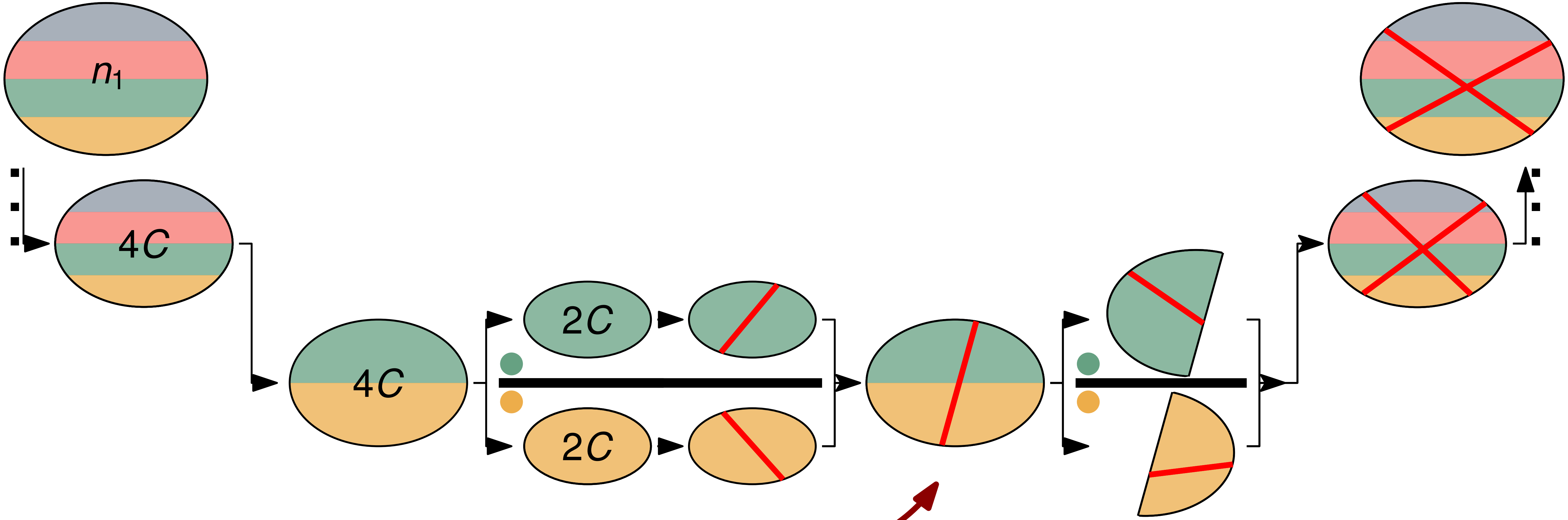
Distributed Deep MGP



invariant: $k_i = \frac{n_i}{C}$

●●●● PEs

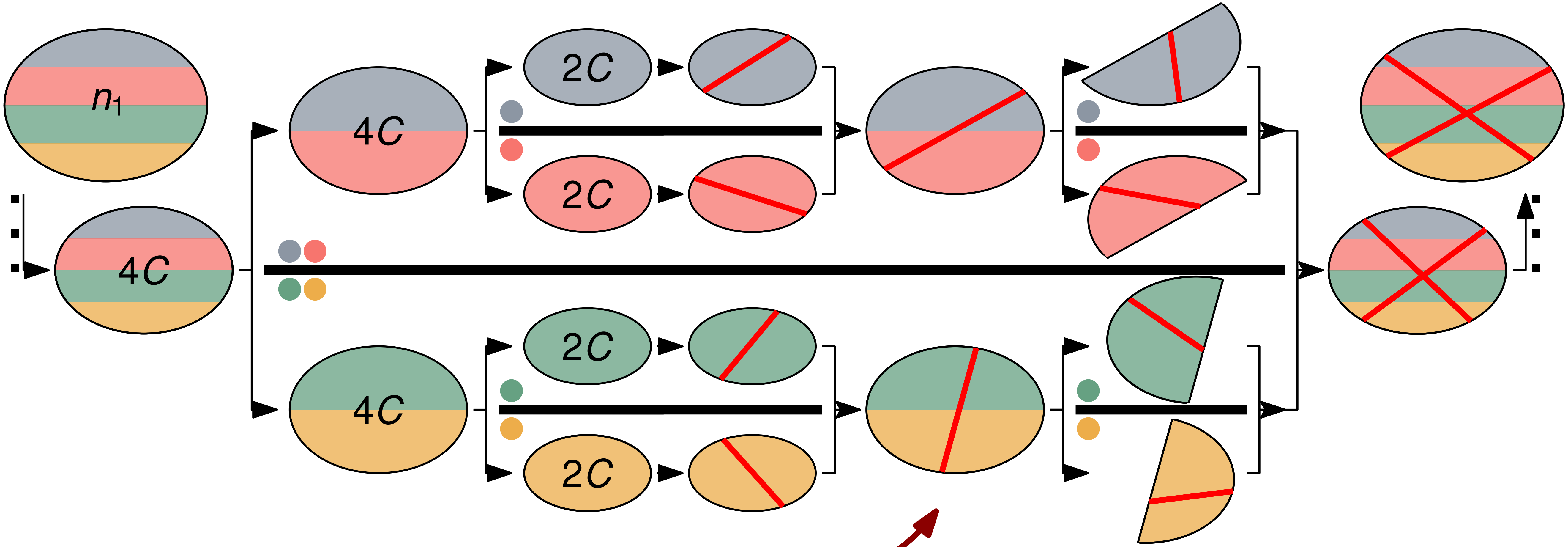
Distributed Deep MGP



invariant: $k_i = \frac{n_i}{C}$

●●●● PEs

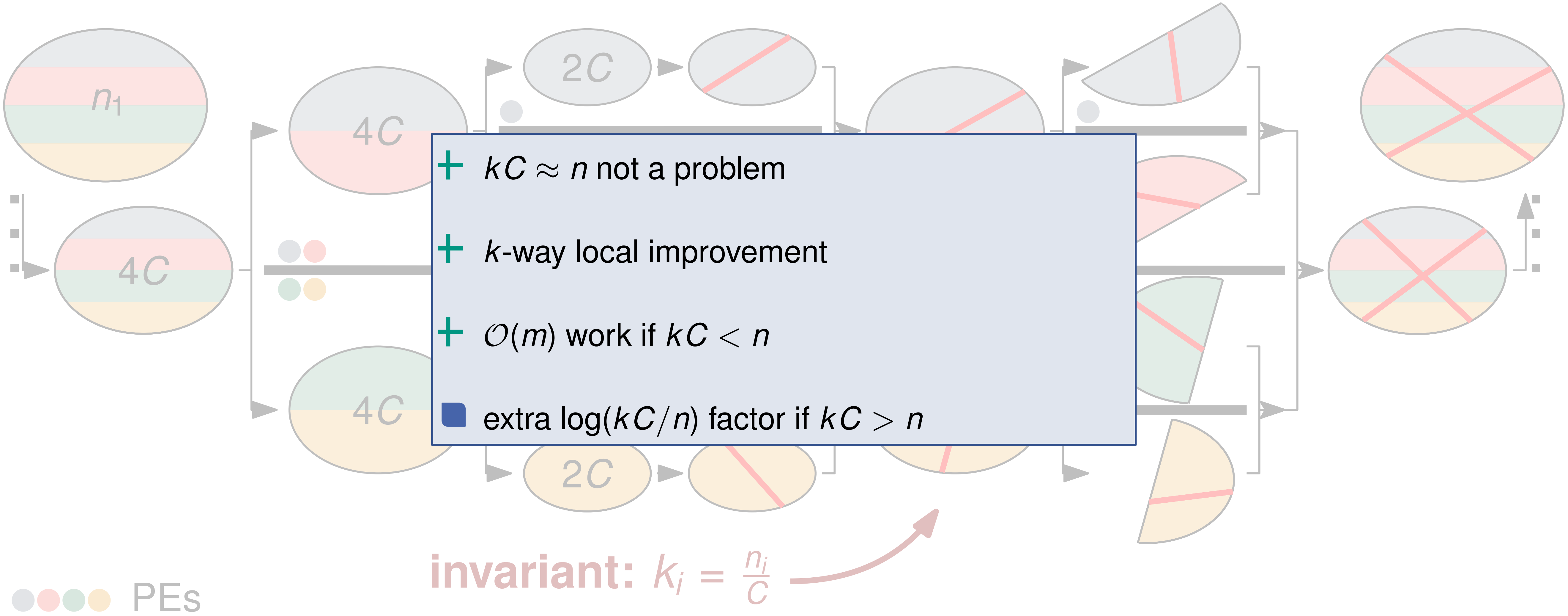
Distributed Deep MGP



invariant: $k_i = \frac{n_i}{C}$

●●●● PEs

Distributed Deep MGP



- + $kC \approx n$ not a problem
- + k-way local improvement
- + $\mathcal{O}(m)$ work if $kC < n$
- ■ extra $\log(kC/n)$ factor if $kC > n$

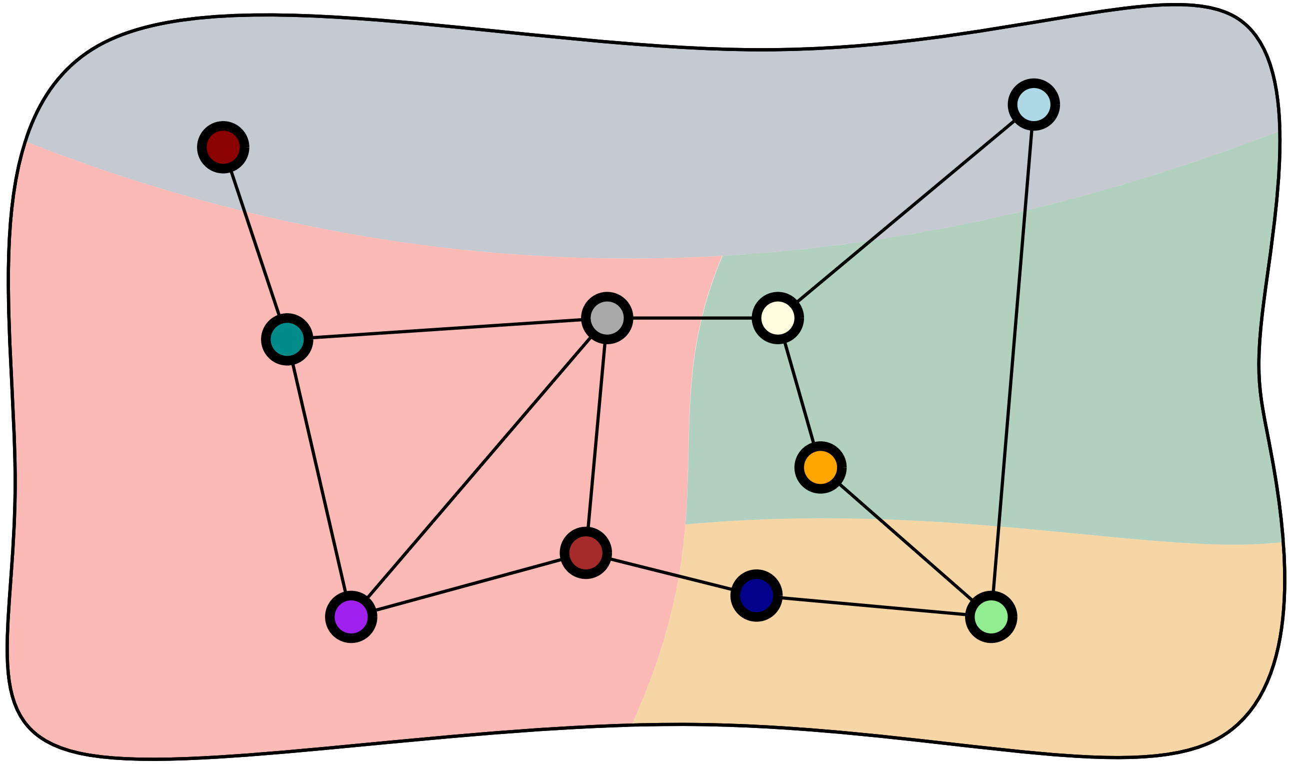
invariant: $k_i = \frac{n_i}{C}$

dKaMinPar: Distributed Deep MGP

- Our contribution: **dKaMinPar** – graph partitioner based on distributed deep MGP
  - Scalable implementation of distributed deep MGP
  - Scalable balanced coarsening and refinement algorithms
  - + many smaller performance improvements over previous works

dKaMinPar – Coarsening

- Coarsening: use size-constrained label propagation: max. weight W

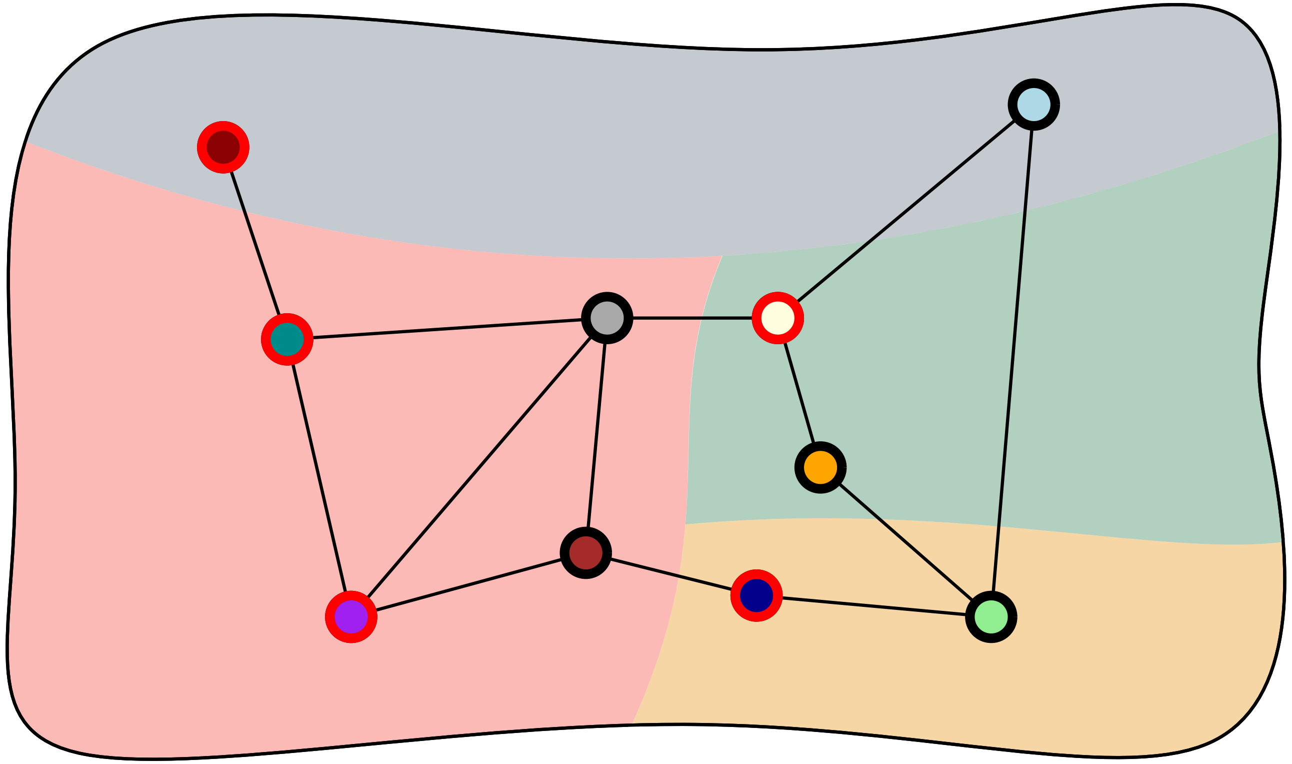


- Constant number of batches

●●●● PEs

dKaMinPar – Coarsening

- Coarsening: use size-constrained label propagation: max. weight *W*

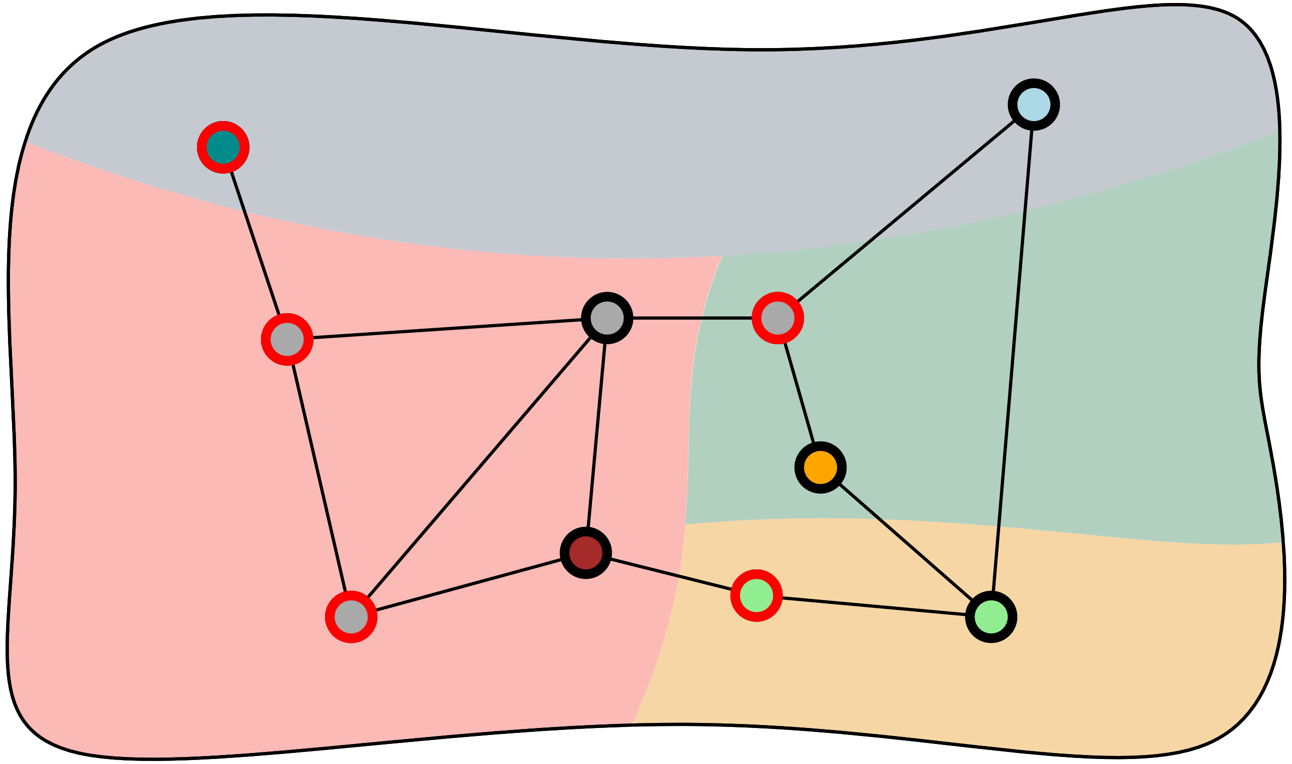


- Constant number of batches
  - Move vertices to adjacent clusters

●●●● PEs

dKaMinPar – Coarsening

- Coarsening: use size-constrained label propagation: max. weight W

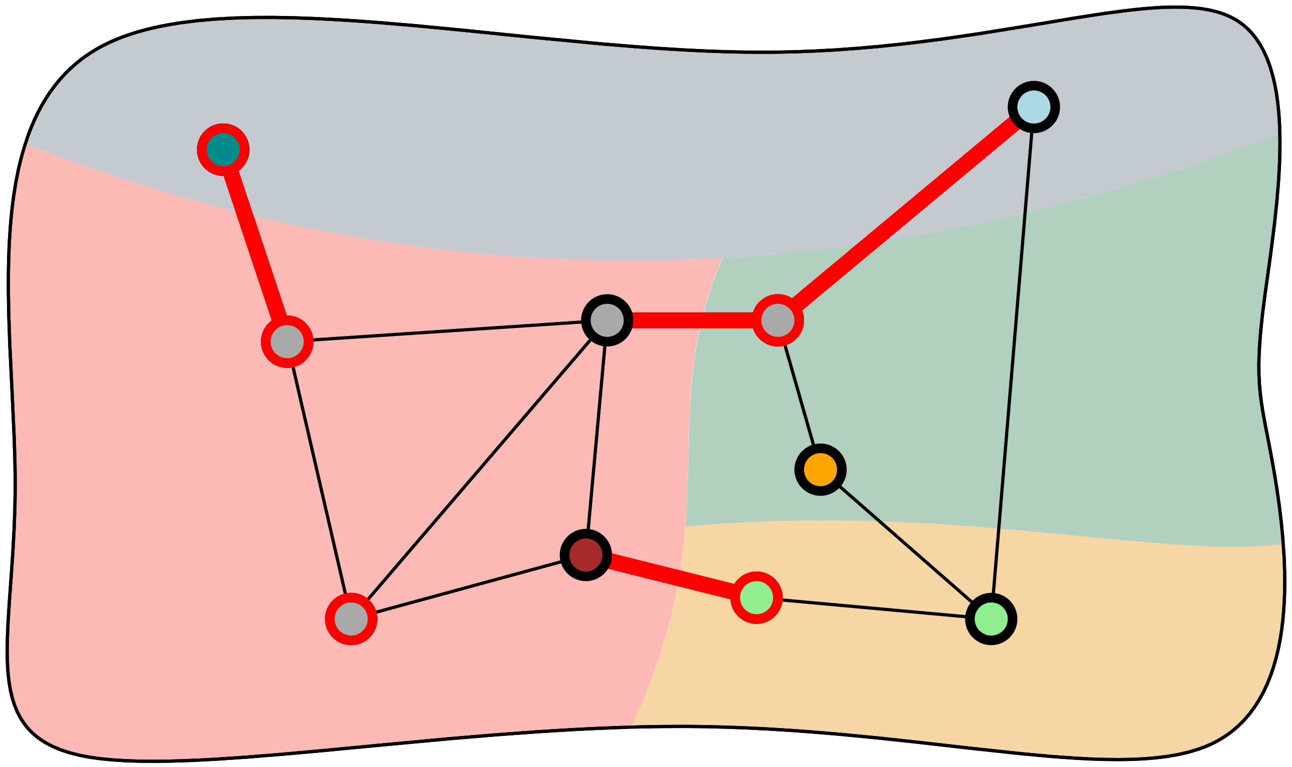


- Constant number of batches
  - Move vertices to adjacent clusters

●●●● PEs

dKaMinPar – Coarsening

- Coarsening: use size-constrained label propagation: max. weight W

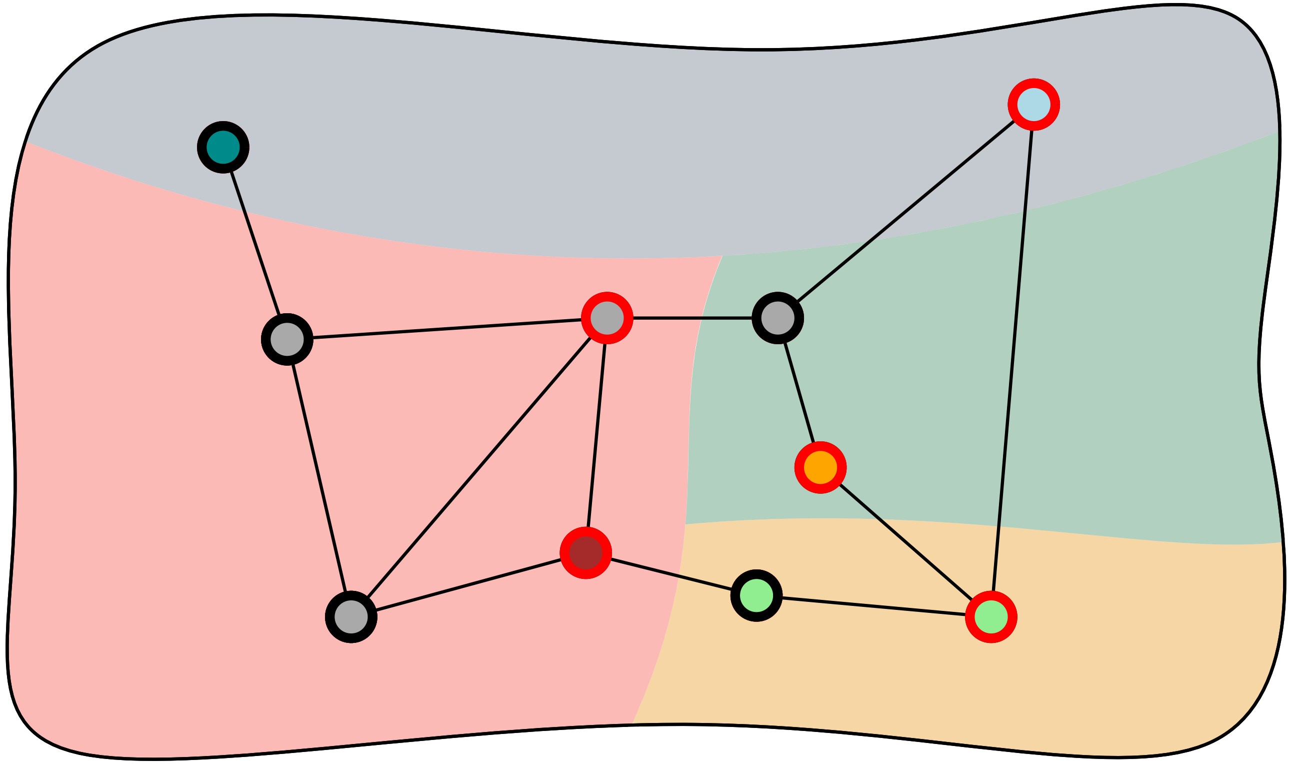


- Constant number of batches
  - Move vertices to adjacent clusters
  - Exchange labels

●●●● PEs

dKaMinPar – Coarsening

- Coarsening: use size-constrained label propagation: max. weight *W*

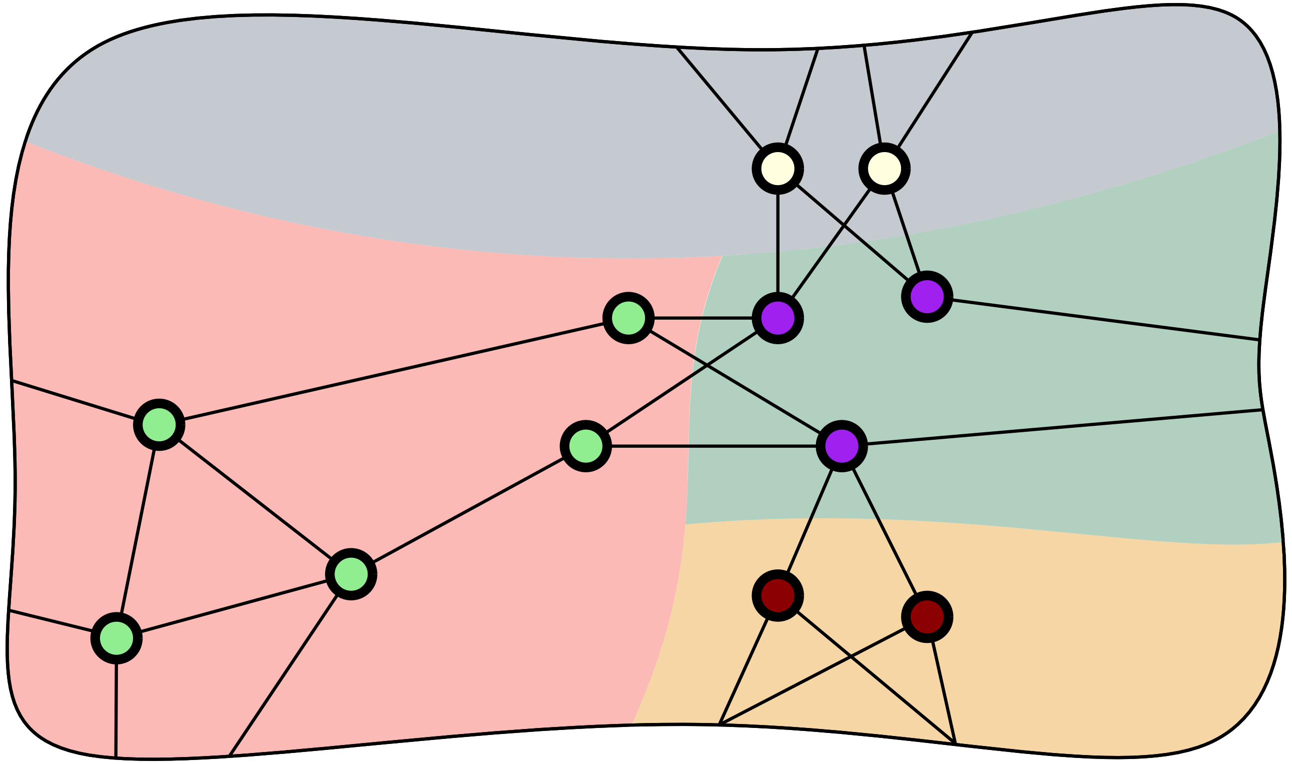


- Constant number of batches
  - Move vertices to adjacent clusters
  - Exchange labels

●●●● PEs

dKaMinPar – Coarsening

- Coarsening: use size-constrained label propagation: max. weight W

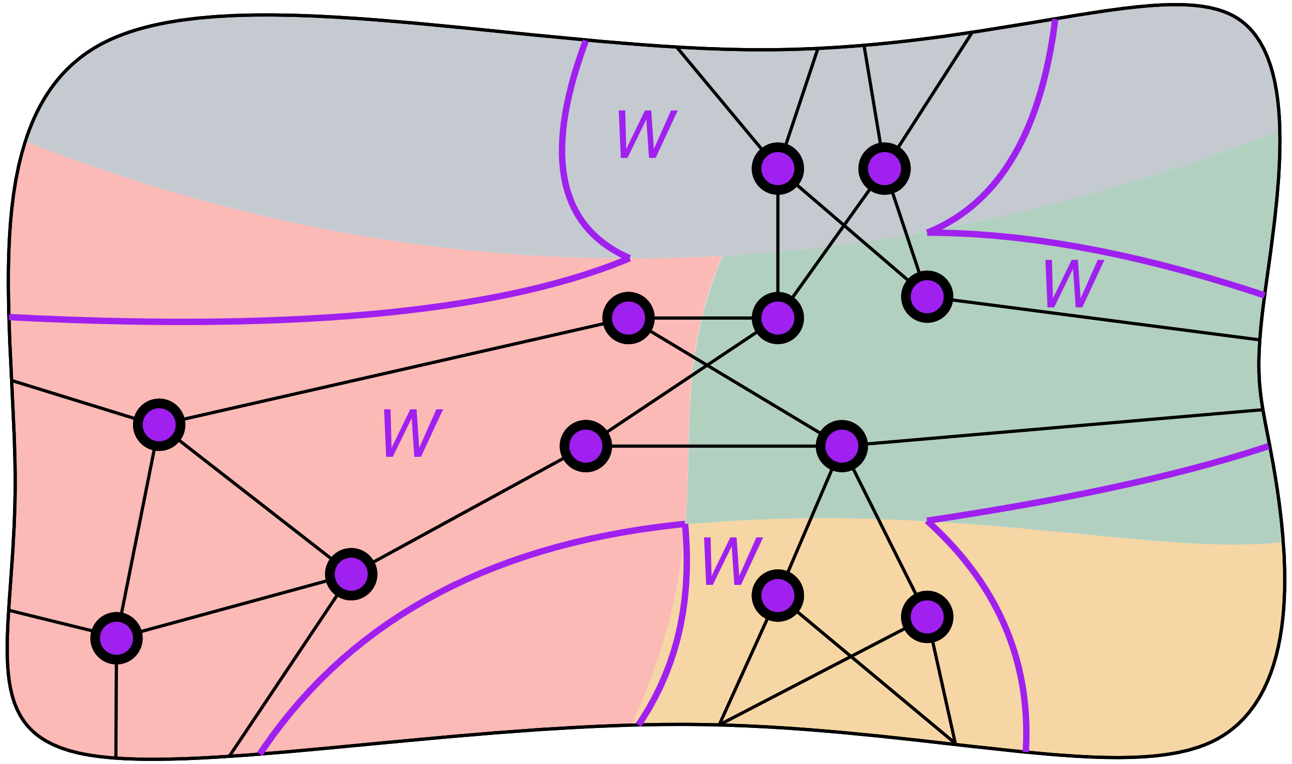


- Constant number of batches
  - Move vertices to adjacent clusters
  - Exchange labels

●●●● PEs

dKaMinPar – Coarsening

- Coarsening: use size-constrained label propagation: max. weight W

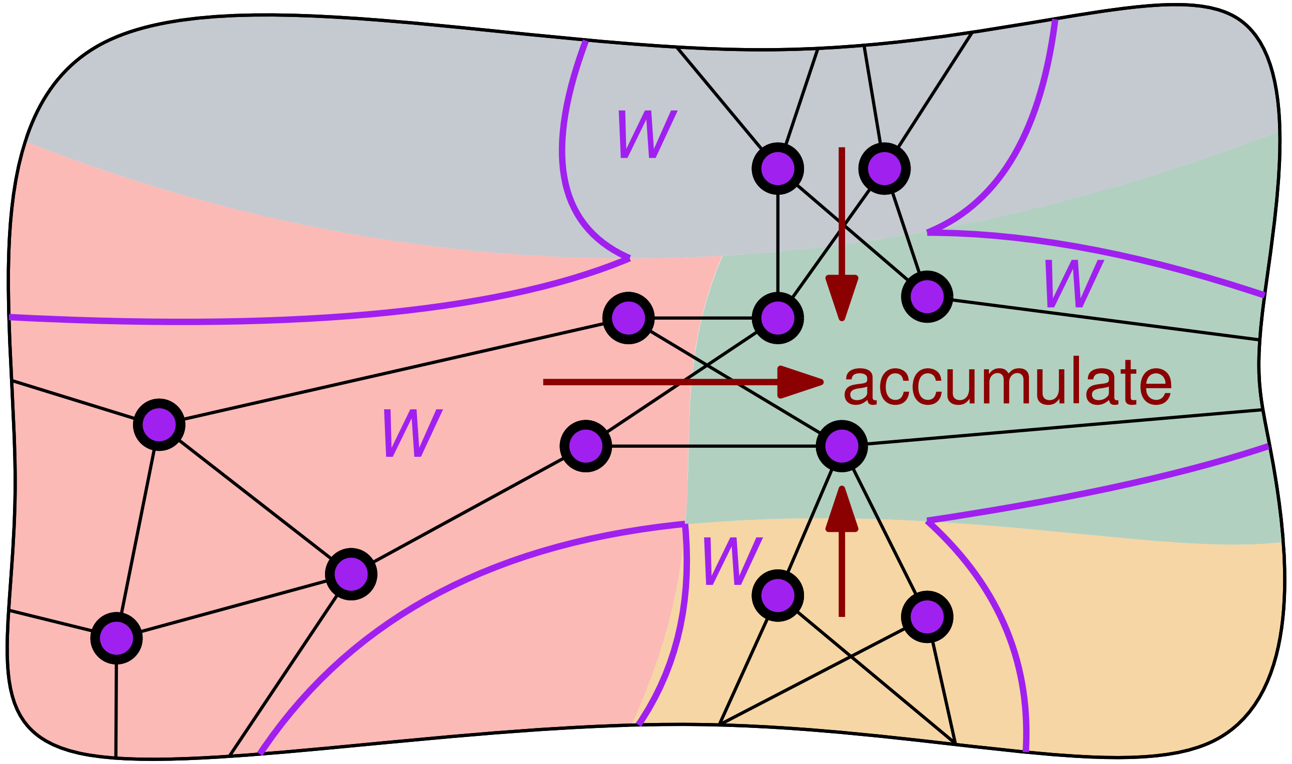


- Constant number of batches
  - Move vertices to adjacent clusters
  - Exchange labels
- Problem: prevent huge clusters

●●●● PEs

dKaMinPar – Coarsening

- Coarsening: use size-constrained label propagation: max. weight W

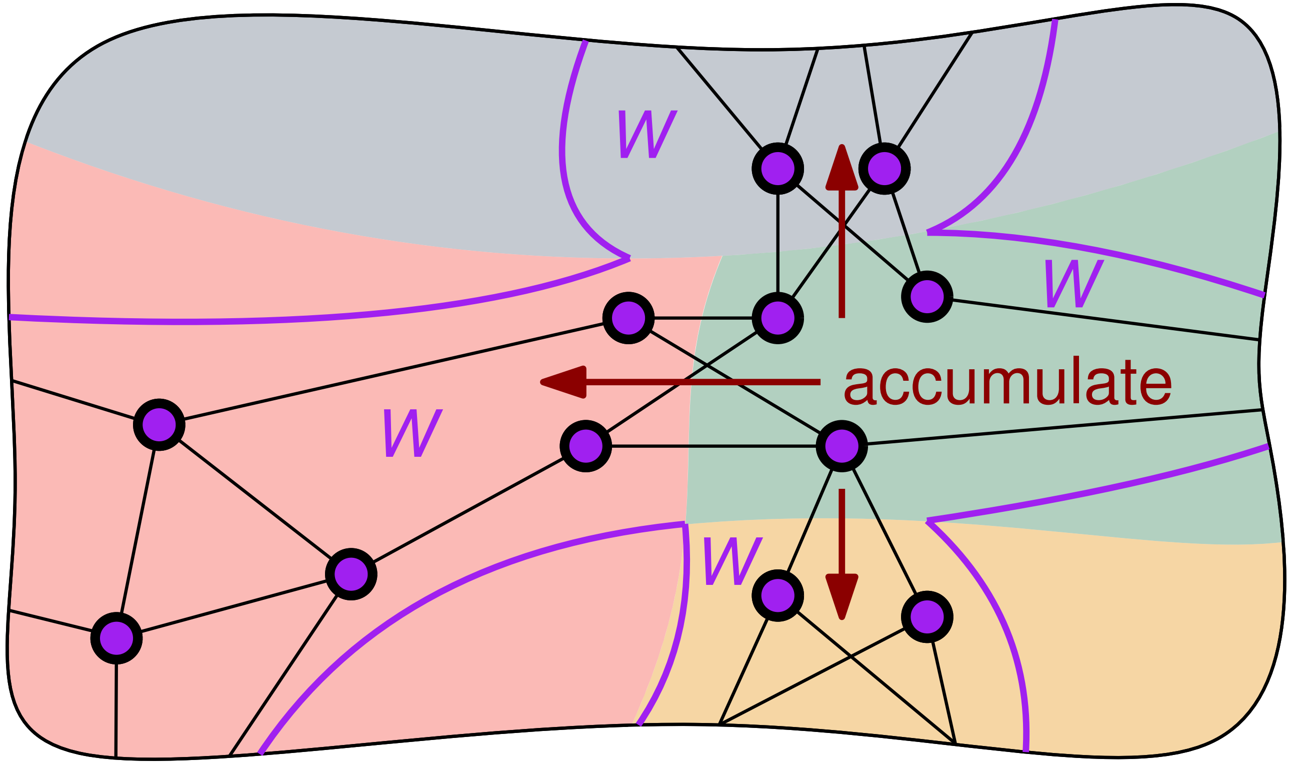


- Constant number of batches
  - Move vertices to adjacent clusters
  - Exchange labels
- Problem: prevent huge clusters

●●●● PEs

dKaMinPar – Coarsening

- Coarsening: use size-constrained label propagation: max. weight W

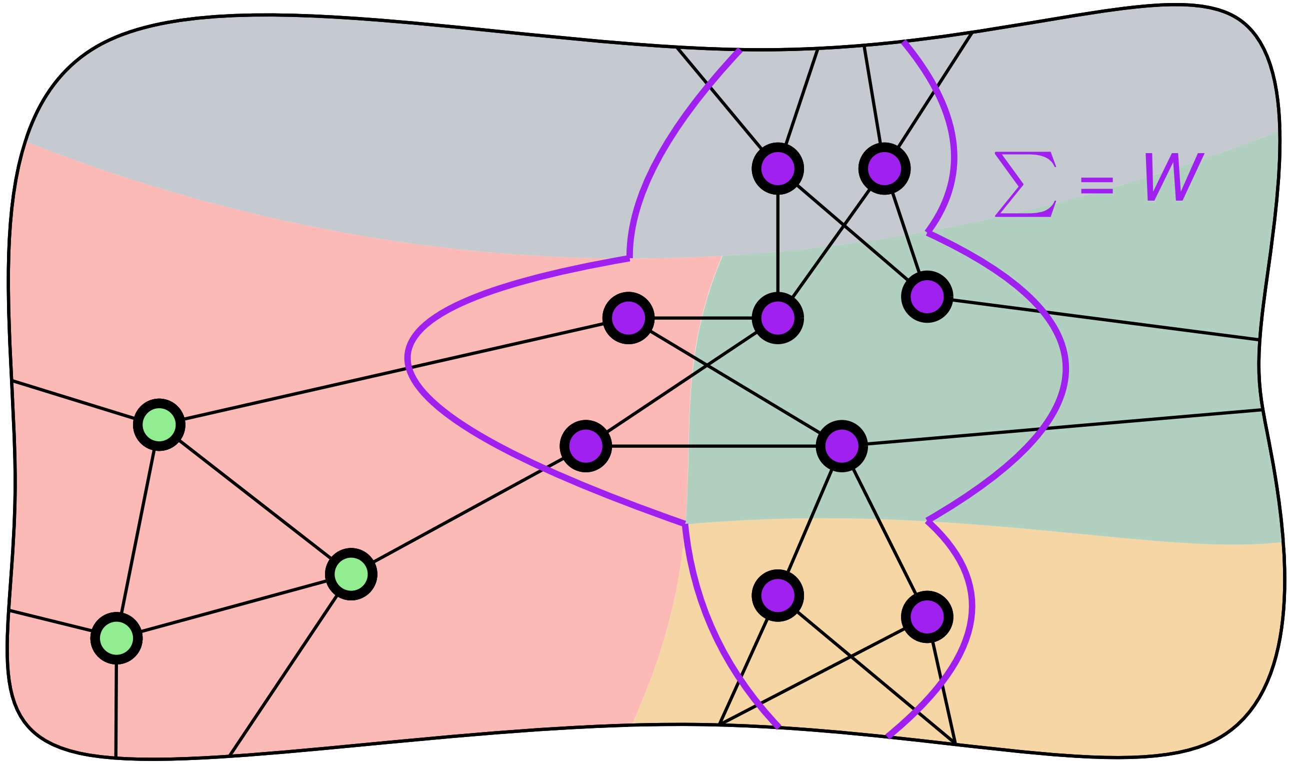


- Constant number of batches
  - Move vertices to adjacent clusters
  - Exchange labels
- Problem: prevent huge clusters

●●●● PEs

dKaMinPar – Coarsening

- Coarsening: use size-constrained label propagation: max. weight W

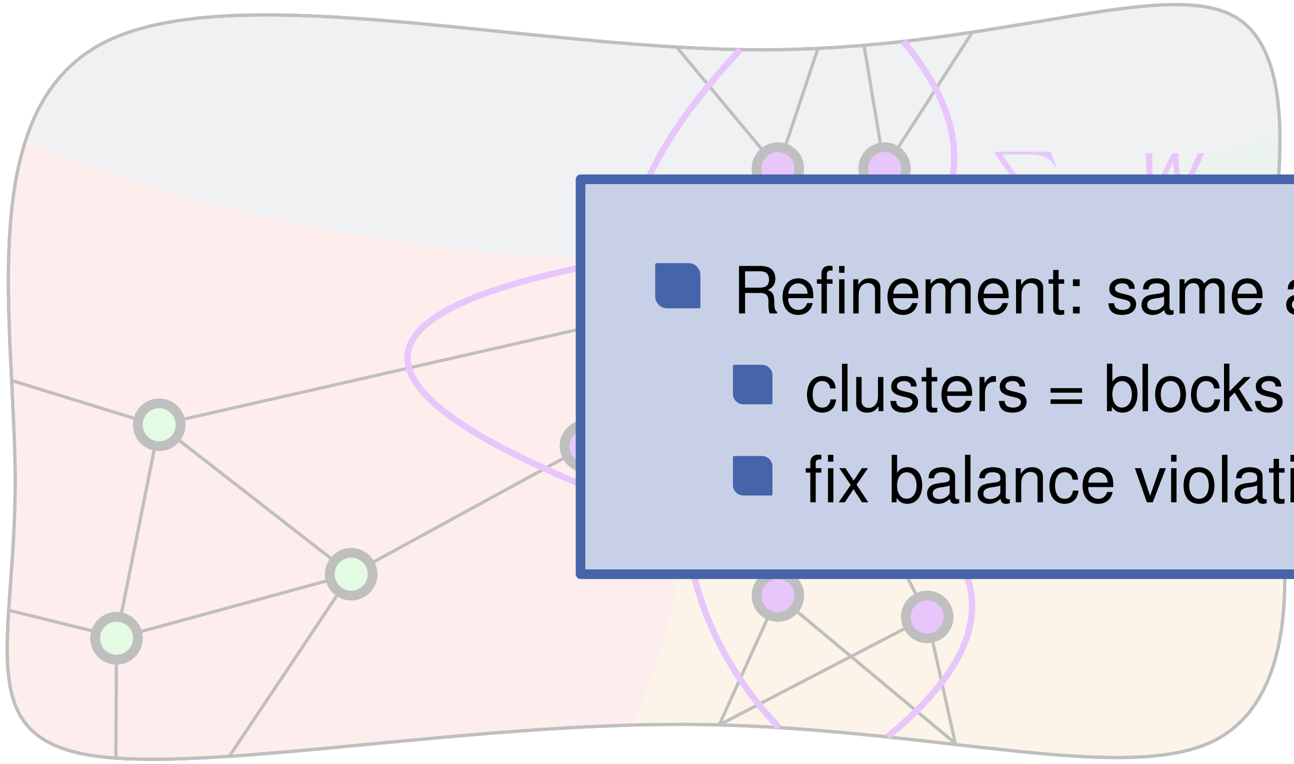


- Constant number of batches
  - Move vertices to adjacent clusters
  - Exchange labels
- Problem: prevent huge clusters
  - Revert label changes prop. to local cluster weight

●●●● PEs

dKaMinPar – Coarsening

- Coarsening: use size-constrained label propagation: max. weight W



●●●● PEs

- Constant number of batches
  - Move vertices to adjacent clusters

    abels

  ent huge clusters

  l changes prop. to local cluster weight

- Refinement: same algorithm
  - clusters = blocks
  - fix balance violations afterwards

Experiments

Experiments – Scalability

- HoreKa: used $\{1, 2, \ldots, 128\}$ nodes $\hat{a}$
  - 2× Intel Xeon Platinum 8368 @ 2.40 GHz
  - 256 GB RAM

- Benchmark sets:
  - weak scaling: rand. geometric + rand. hyperbolic graphs
  - strong scaling: rand. geometric + 5 real world graphs

- Comparing **dKaMinPar** against:
  - ParHiP
  - ParMETIS
  - (XtraPuLP)

Experiments – Scalability

- HoreKa: used $\{1, 2, \ldots, 128\}$ nodes *â*
  - 2× Intel Xeon Platinum 8368 @ 2.40 GHz (used 64 out of 78 cores)
  - 256 GB RAM

- Benchmark sets:
  - weak scaling: rand. geometric + rand. hyperbolic graphs
  - strong scaling: rand. geometric + 5 real world graphs

- Comparing **dKaMinPar** against:
  - ParHiP
  - ParMETIS
  - (XtraPuLP)

Experiments – Scalability

- HoreKa: used $\{1, 2, \ldots, 128\}$ nodes â
  - $2\times$ Intel Xeon Platinum 8368 @ 2.40 GHz (used 64 out of 78 cores)
  - 256 GB RAM

- Benchmark sets: up to 2^{39} edges on 2^{13} cores
  - weak scaling: rand. geometric + rand. hyperbolic graphs
  - strong scaling: rand. geometric + 5 real world graphs

- Comparing **dKaMinPar** against:
  - ParHiP
  - ParMETIS
  - (XtraPuLP)

Experiments – Scalability

- HoreKa: used $\{1, 2, \ldots, 128\}$ nodes â
  - 2× Intel Xeon Platinum 8368 @ 2.40 GHz (used 64 out of 78 cores)
  - 256 GB RAM
- Benchmark sets: up to 2^{39} edges on 2^{13} cores
  - weak scaling: rand. geometric + rand. hyperbolic graphs
  - strong scaling: rand. geometric + 5 real world graphs 3× ”irregular”, 3× ”regular”
- Comparing **dKaMinPar** against:
  - ParHiP
  - ParMETIS
  - (XtraPuLP)

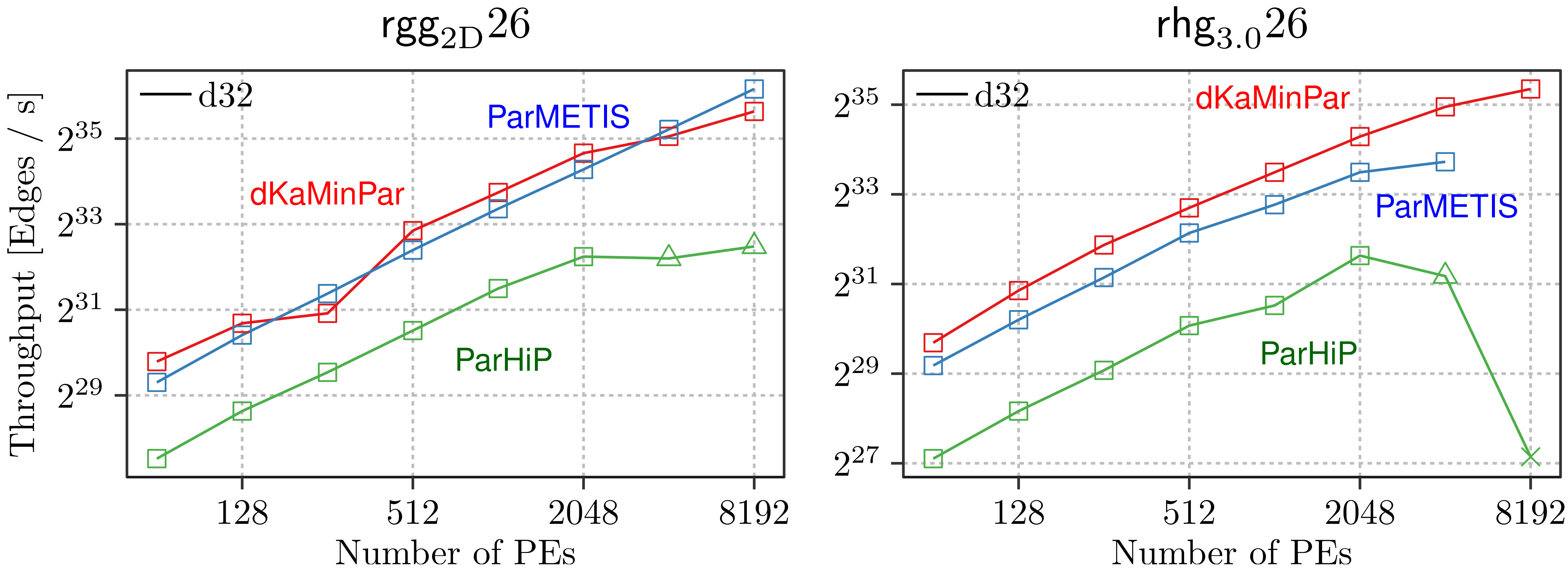
Experiments – Scalability

- HoreKa: used $\{1, 2, \ldots, 128\}$ nodes â
  - 2× Intel Xeon Platinum 8368 @ 2.40 GHz (used 64 out of 78 cores)
  - 256 GB RAM
- Benchmark sets: up to 2^{39} edges on 2^{13} cores
  - weak scaling: rand. geometric + rand. hyperbolic graphs
  - strong scaling: rand. geometric + 5 real world graphs 3× ”irregular”, 3× ”regular”
- Comparing **dKaMinPar** against:
  - ParHiP multilevel, label propagation for coarsening + refinement
  - ParMETIS
  - (XtraPuLP)

Experiments – Scalability

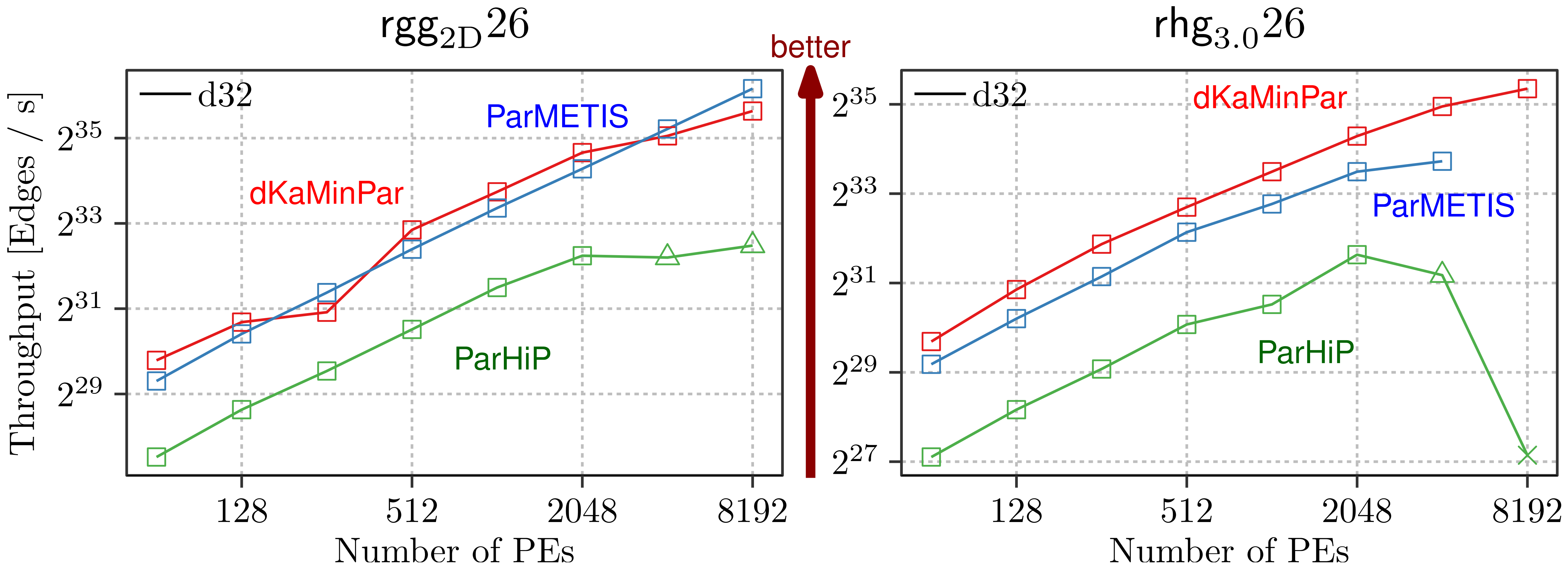
- HoreKa: used $\{1, 2, \ldots, 128\}$ nodes â
  - 2× Intel Xeon Platinum 8368 @ 2.40 GHz (used 64 out of 78 cores)
  - 256 GB RAM
- Benchmark sets: up to 2^{39} edges on 2^{13} cores
  - weak scaling: rand. geometric + rand. hyperbolic graphs
  - strong scaling: rand. geometric + 5 real world graphs 3× ”irregular”, 3× ”regular”
- Comparing **dKaMinPar** against:
  - ParHiP multilevel, label propagation for coarsening + refinement
  - ParMETIS multilevel, matchings for coarsening, greedy refinement
  - (XtraPuLP)

Experiments – Weak Scaling, constant $k = 16$



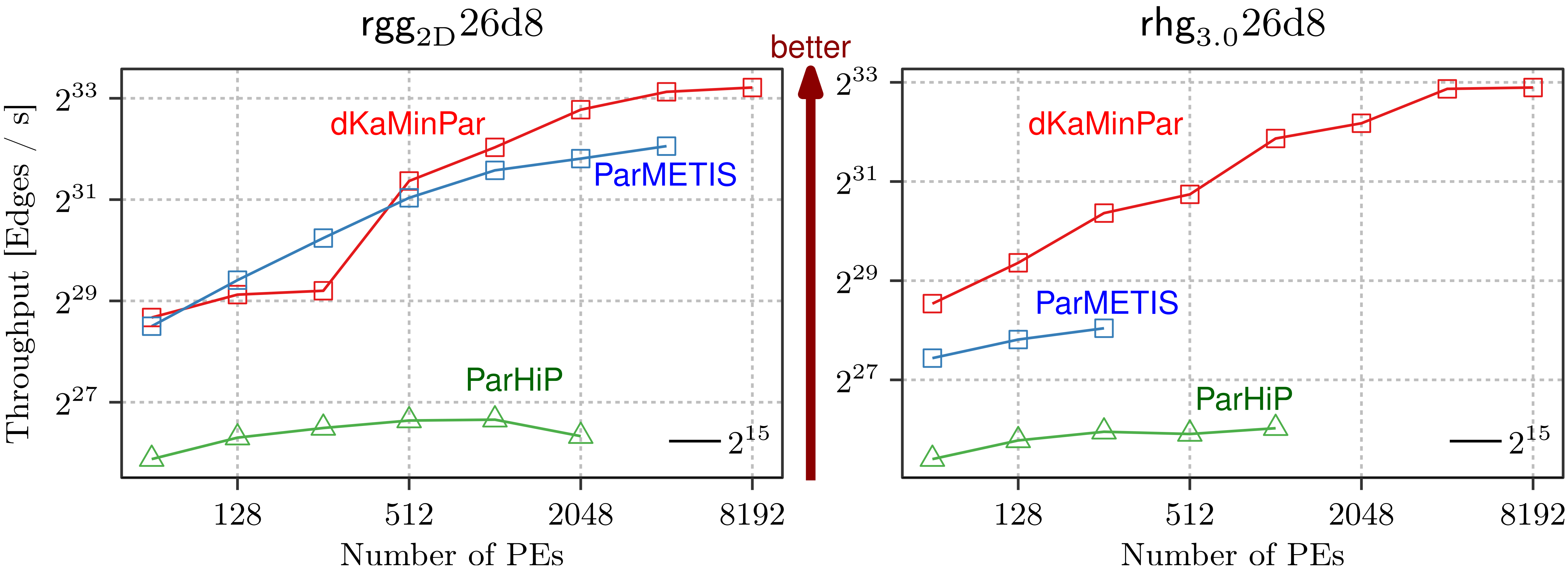
[$\{1, 2, \ldots, 128\}$ nodes @ 64 cores]

Experiments – Weak Scaling, constant $k = 16$



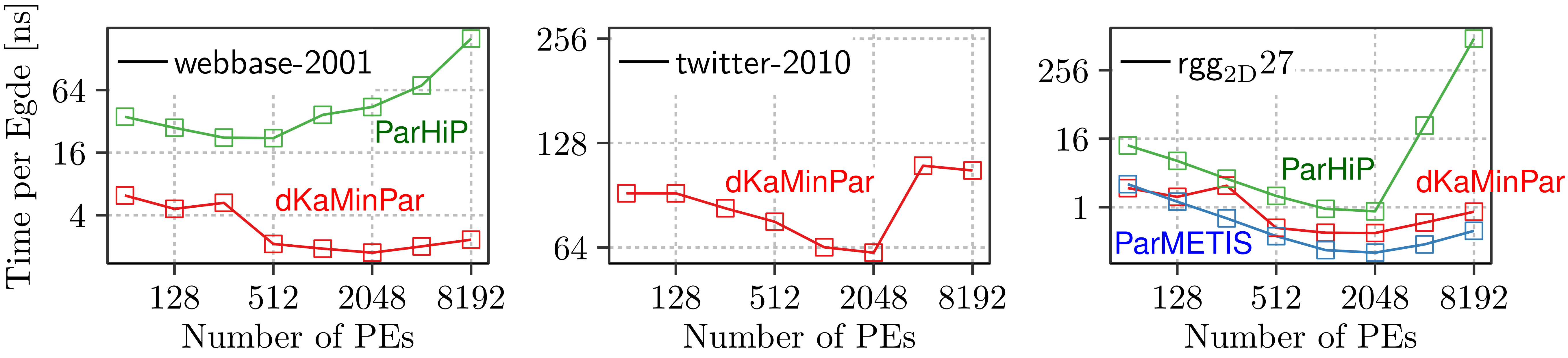
[$\{1, 2, \ldots, 128\}$ nodes @ 64 cores]

Experiments – Weak Scaling, constant $\frac{n}{k} = 2^{15}$



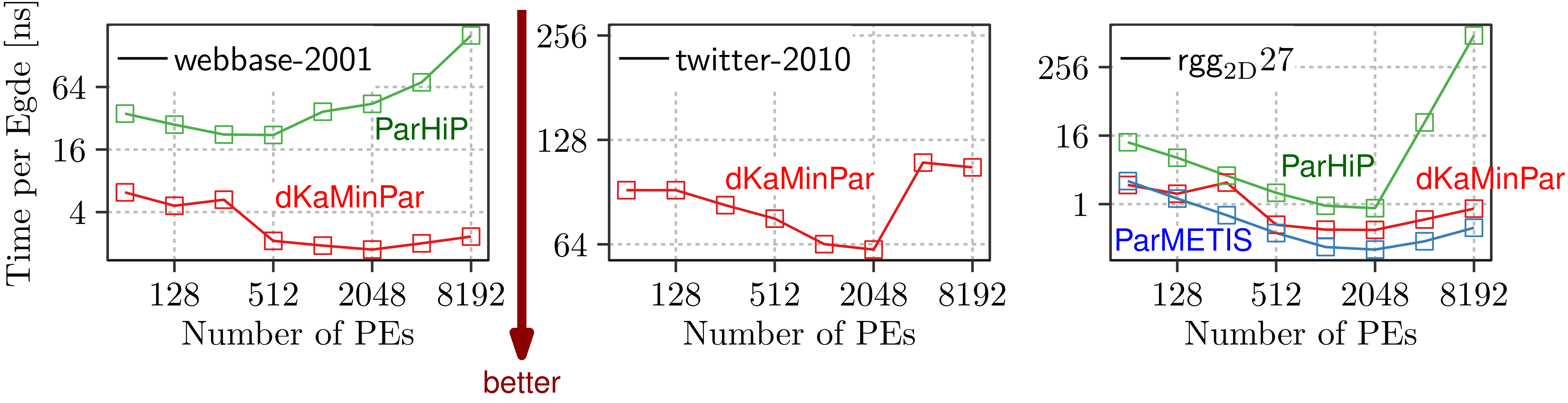
[{1, 2, . . . , 128} nodes @ 64 cores]

Experiments – Strong Scaling



[$\{1, 2, \ldots, 128\}$ nodes @ 64 cores]

Experiments – Strong Scaling



[$\{1, 2, \ldots, 128\}$ **nodes @ 64 cores]**

Experiments – Scalability: Quality

Graph	Cut on 64 PEs	Cut on 8192 PEs
kmer_V1r	10 955	10 836
nlpkkt240	5 726	5 547
rgg27	353	347
webbase-2001	9 674	9 524
uk-2007-05	4 054	4 064
twitter-2010	616 791	588 380

×1 000

Experiments – Scalability: Quality

Graph	Cut on 64 PEs	Cut on 8192 PEs
kmer_V1r	10 955	10 836
nlpkkt240	5 726	5 547
rgg27	353	347
webbase-2001	9 674	9 524
uk-2007-05	4 054	4 064
twitter-2010	616 791	588 380

no → degradation

×1 000

Experiments – Quality

- 64 cores of 1 AMD EPYC 7702 @ 2 GHz, 1 TB RAM

- Benchmark set: 32 graphs with $4.7\,\mathrm{M} \leq \mathrm{m} \leq 6.6\,\mathrm{G}$
- $\varepsilon = 3\%$
- $k \in \{2, 4, \ldots, 128\}$

- Comparing **dKaMinPar** against:

Distributed-mem.

- ParHiP
- ParMETIS

Shared-mem.

- KaMinPar

Experiments – Quality

- 64 cores of 1 AMD EPYC 7702 @ 2 GHz, 1 TB RAM

- Benchmark set: 32 graphs with $4.7\,\mathrm{M} \leq m \leq 6.6\,\mathrm{G}$ $\begin{cases} 17 \times \text{regular, } 3 \leq \Delta \leq 40 \\ 15 \times \text{irregular, } 2.7\,\mathrm{k} \leq \Delta \leq 8.6\,\mathrm{M} \end{cases}$
- $\varepsilon = 3\%$
- $k \in \{2, 4, \ldots, 128\}$

- Comparing **dKaMinPar** against:

Distributed-mem.

- ParHiP
- ParMETIS

Shared-mem.

- KaMinPar

Experiments – Quality

- 64 cores of 1 AMD EPYC 7702 @ 2 GHz, 1 TB RAM

- Benchmark set: 32 graphs with $4.7\,\mathrm{M} \leq m \leq 6.6\,\mathrm{G}$ $\begin{cases} 17 \times \text{regular}, 3 \leq \Delta \leq 40 \\ 15 \times \text{irregular}, 2.7\,\text{k} \leq \Delta \leq 8.6\,\text{M} \end{cases}$
- $\varepsilon = 3\%$
- $k \in \{2, 4, \ldots, 128\}$

- Comparing **dKaMinPar** against:

Distributed-mem.

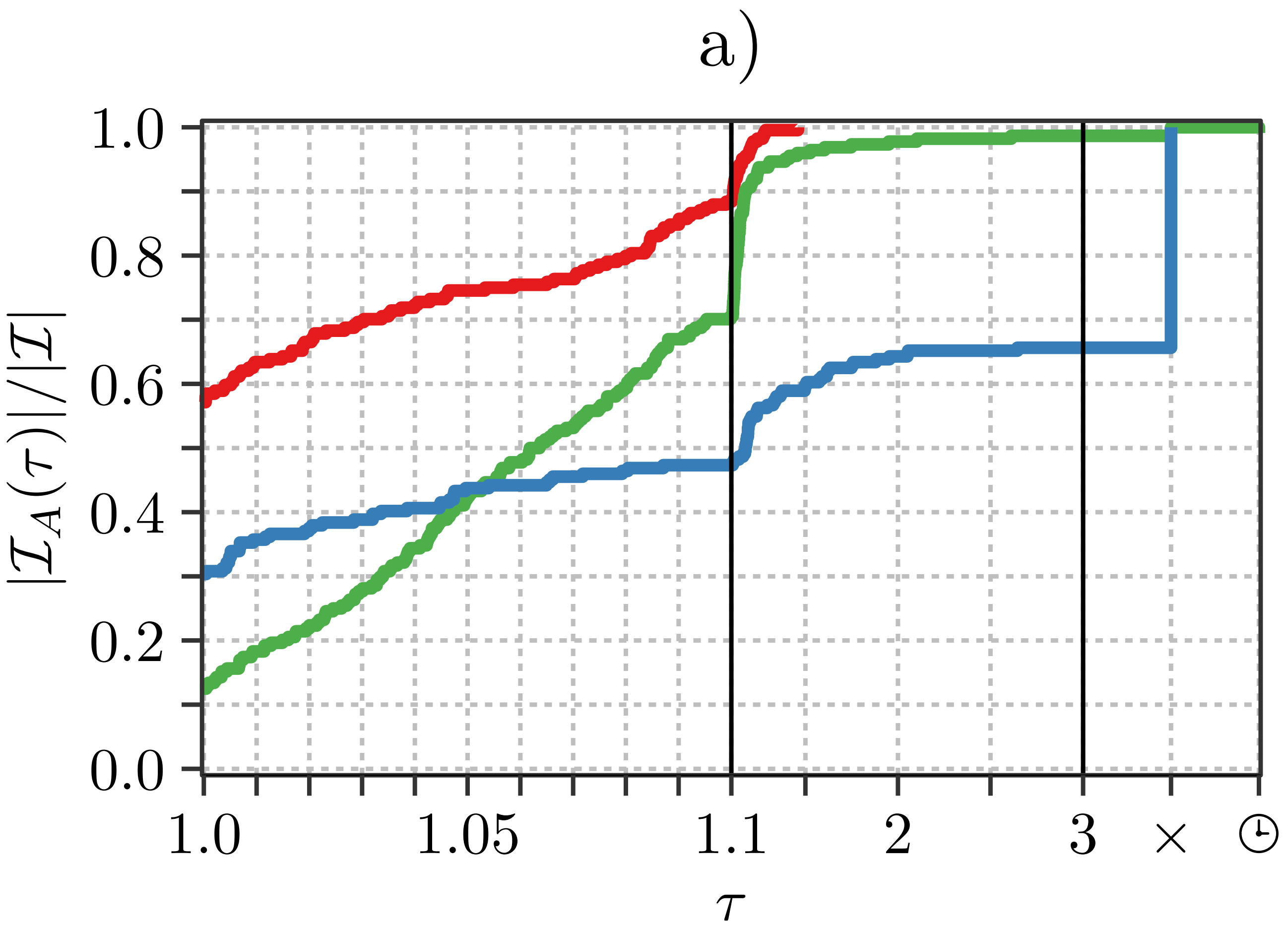
  - ParHiP
  - ParMETIS

Shared-mem.

  - KaMinPar

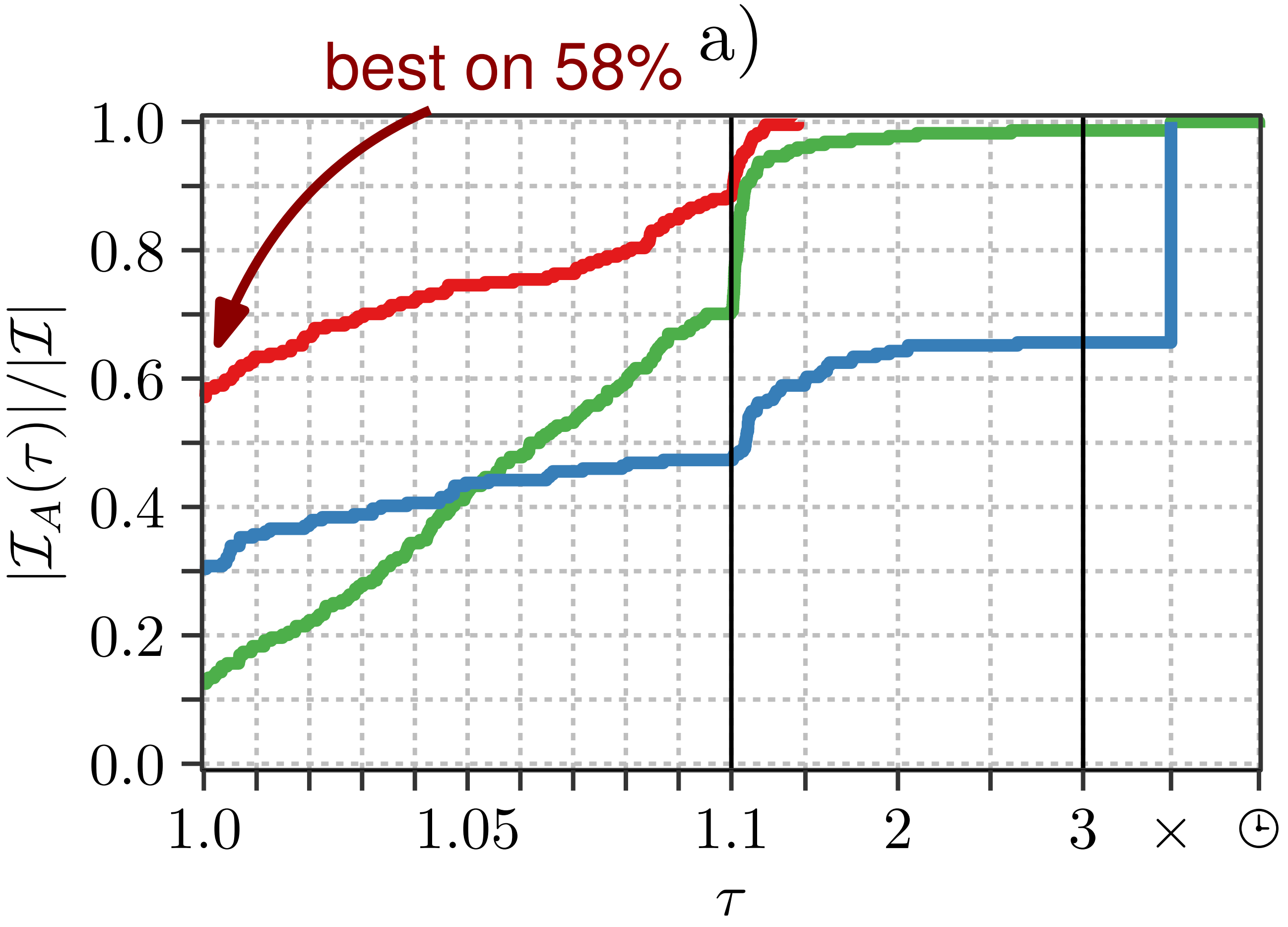
similar techniques implemented in shared-memory

Experiments – Quality: vs. Competitors



[1 node @ 64 cores]

Experiments – Quality: vs. Competitors

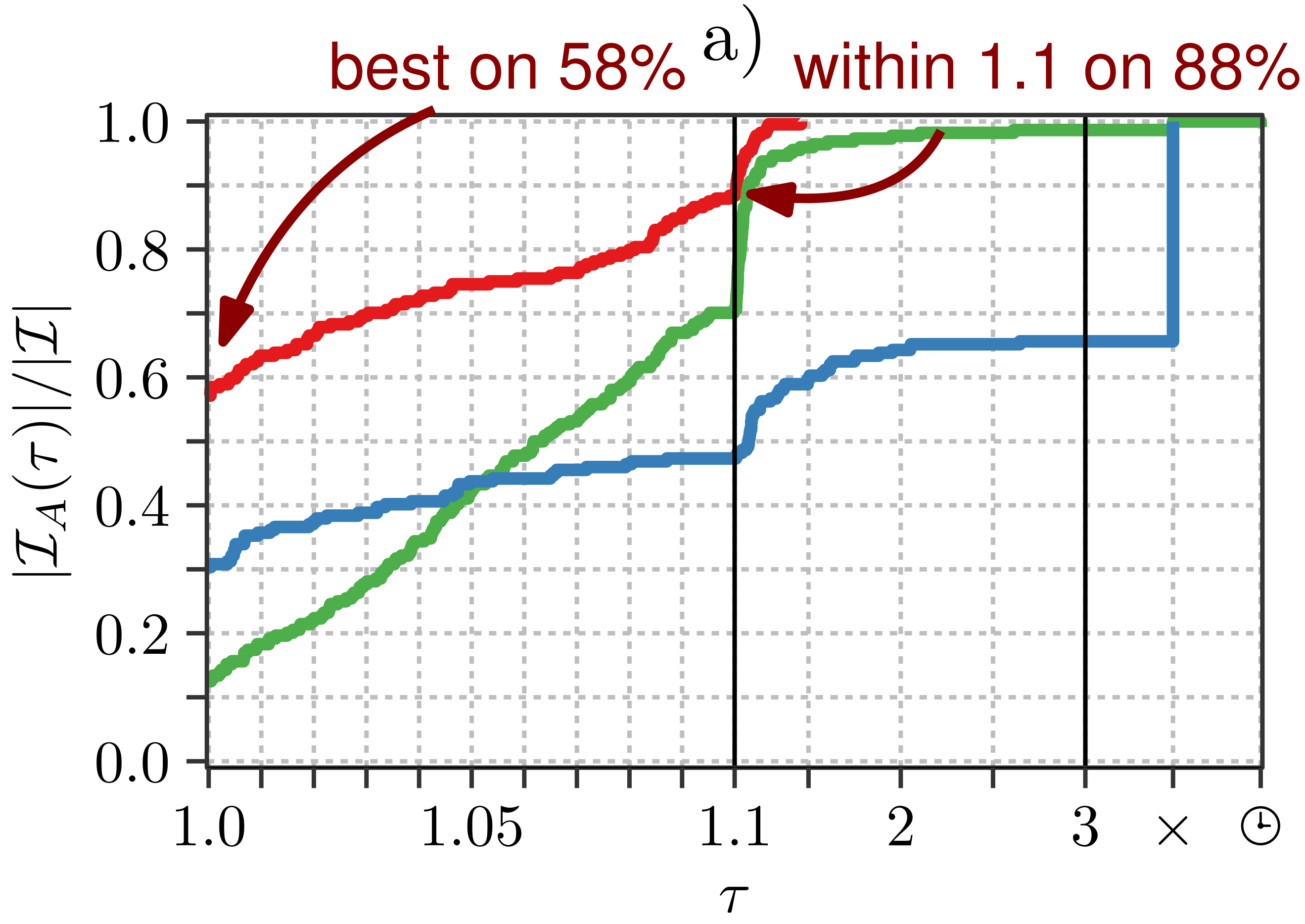


[1 node @ 64 cores]

Experiments – Quality: vs. Competitors

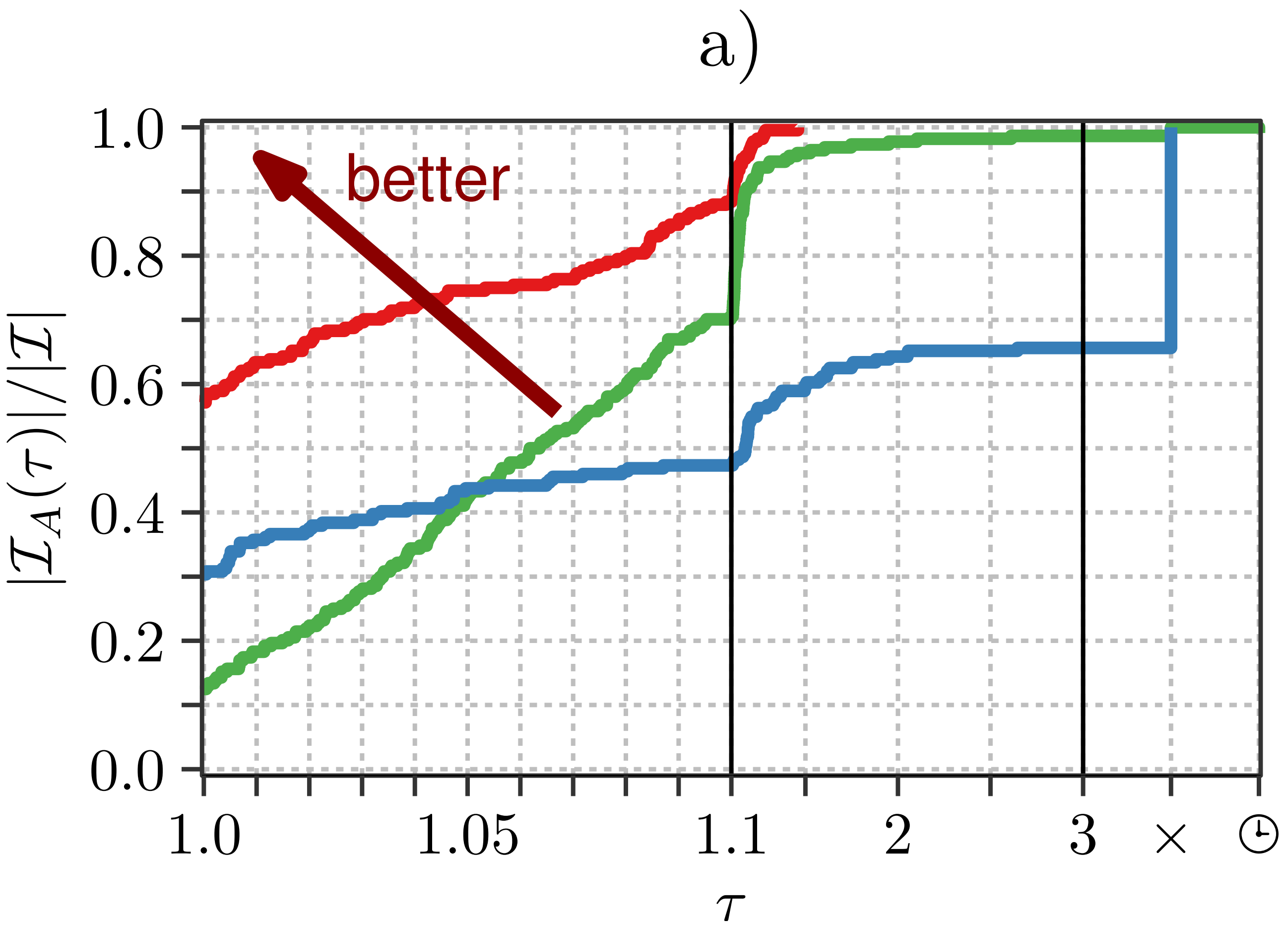
best on 58%

within 1.1 on 88%



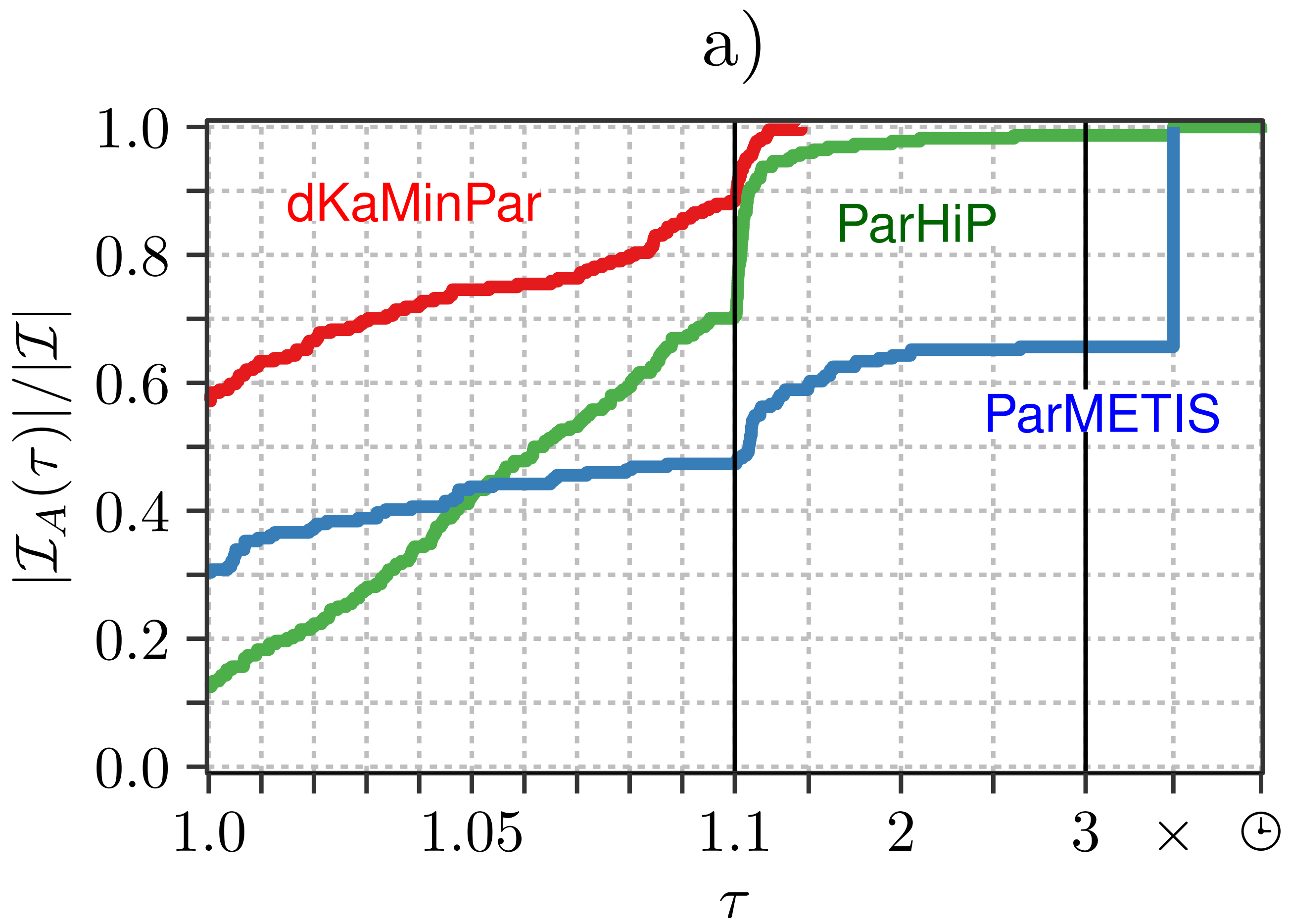
[1 node @ 64 cores]

Experiments – Quality: vs. Competitors



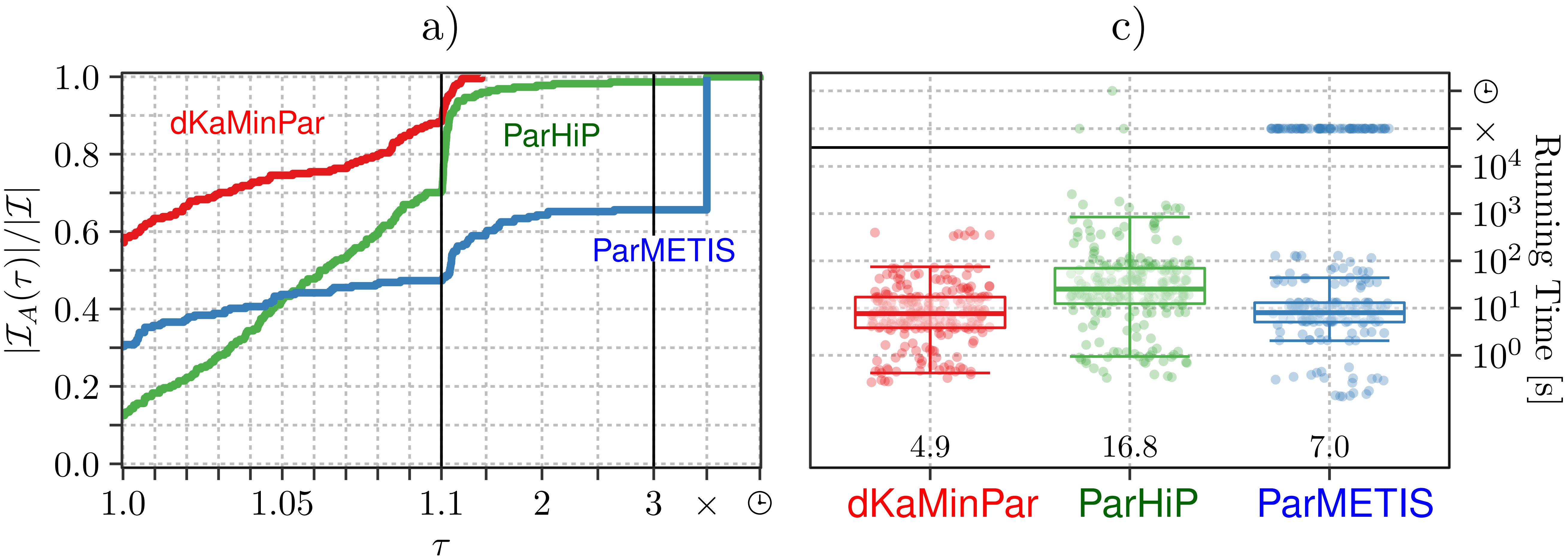
[1 node @ 64 cores]

Experiments – Quality: vs. Competitors



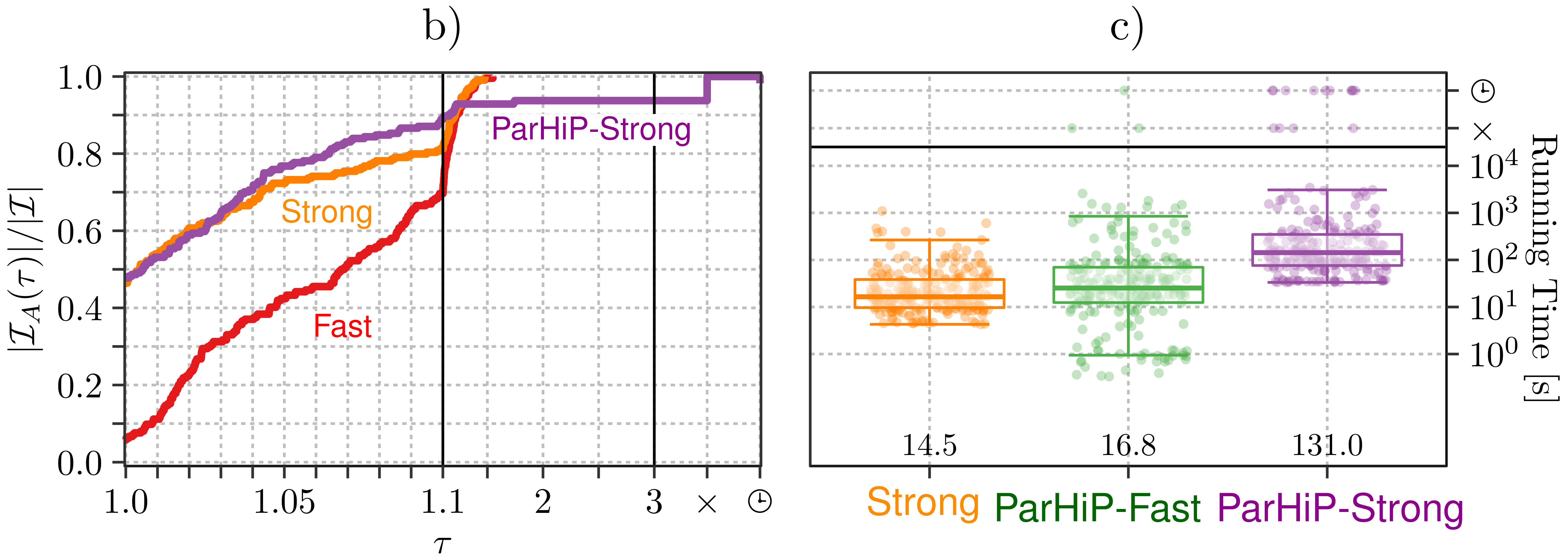
[1 node @ 64 cores]

Experiments – Quality: vs. Competitors



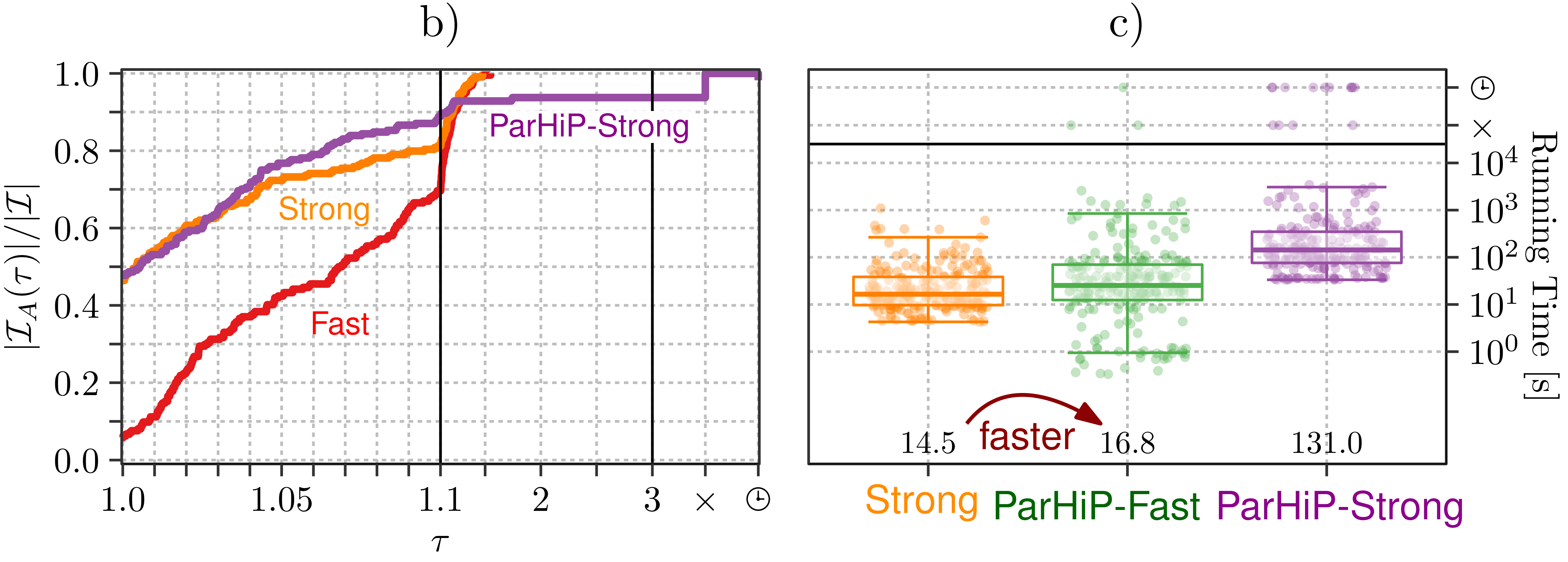
[1 node @ 64 cores]

Experiments – Quality: vs. Competitors



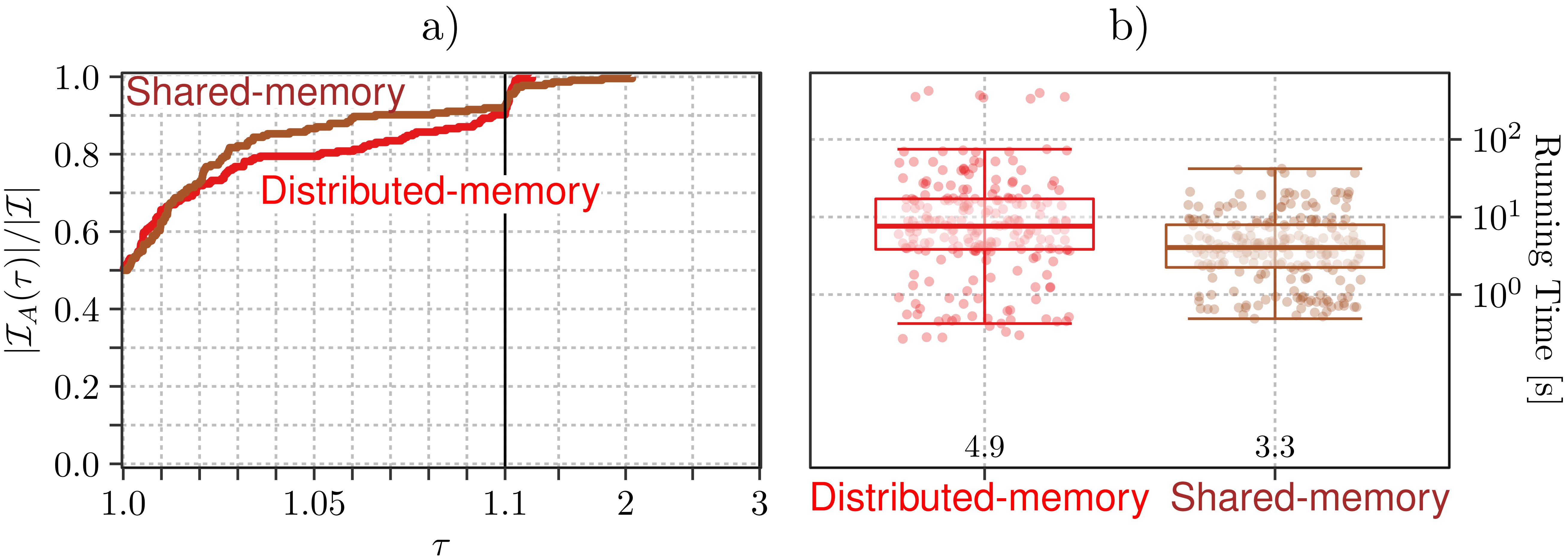
[1 node @ 64 cores]

Experiments – Quality: vs. Competitors



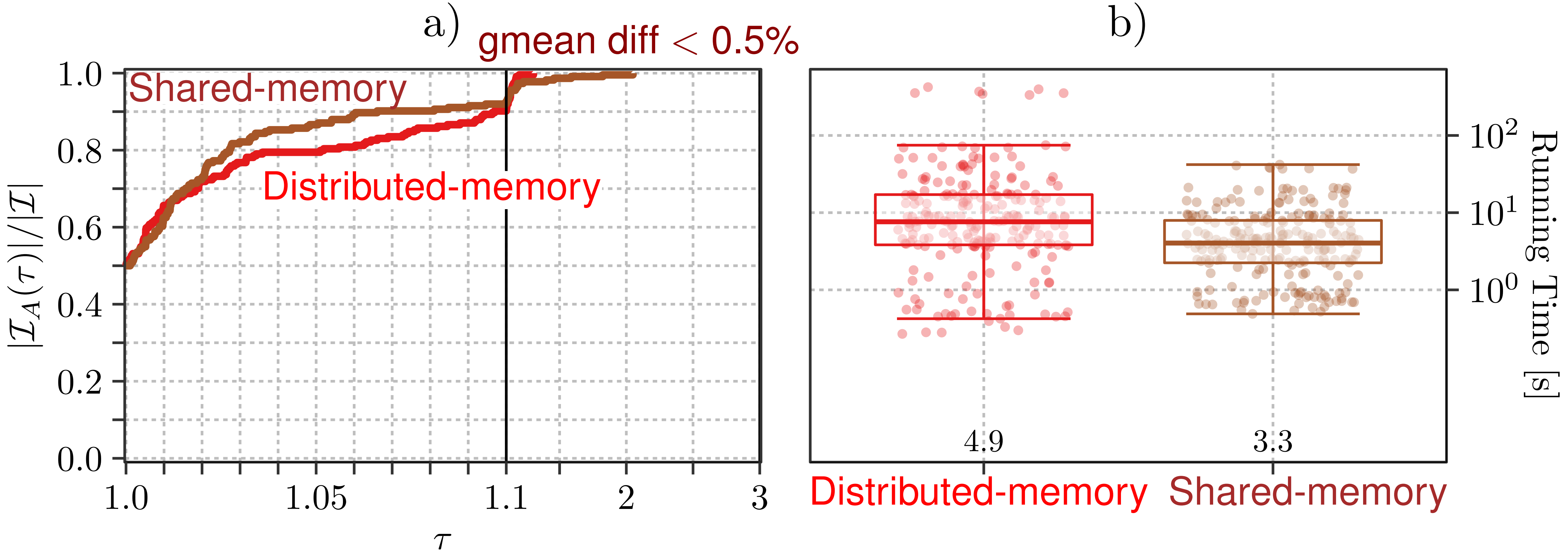
[1 node @ 64 cores]

Experiments – Quality: vs. Shared-Memory



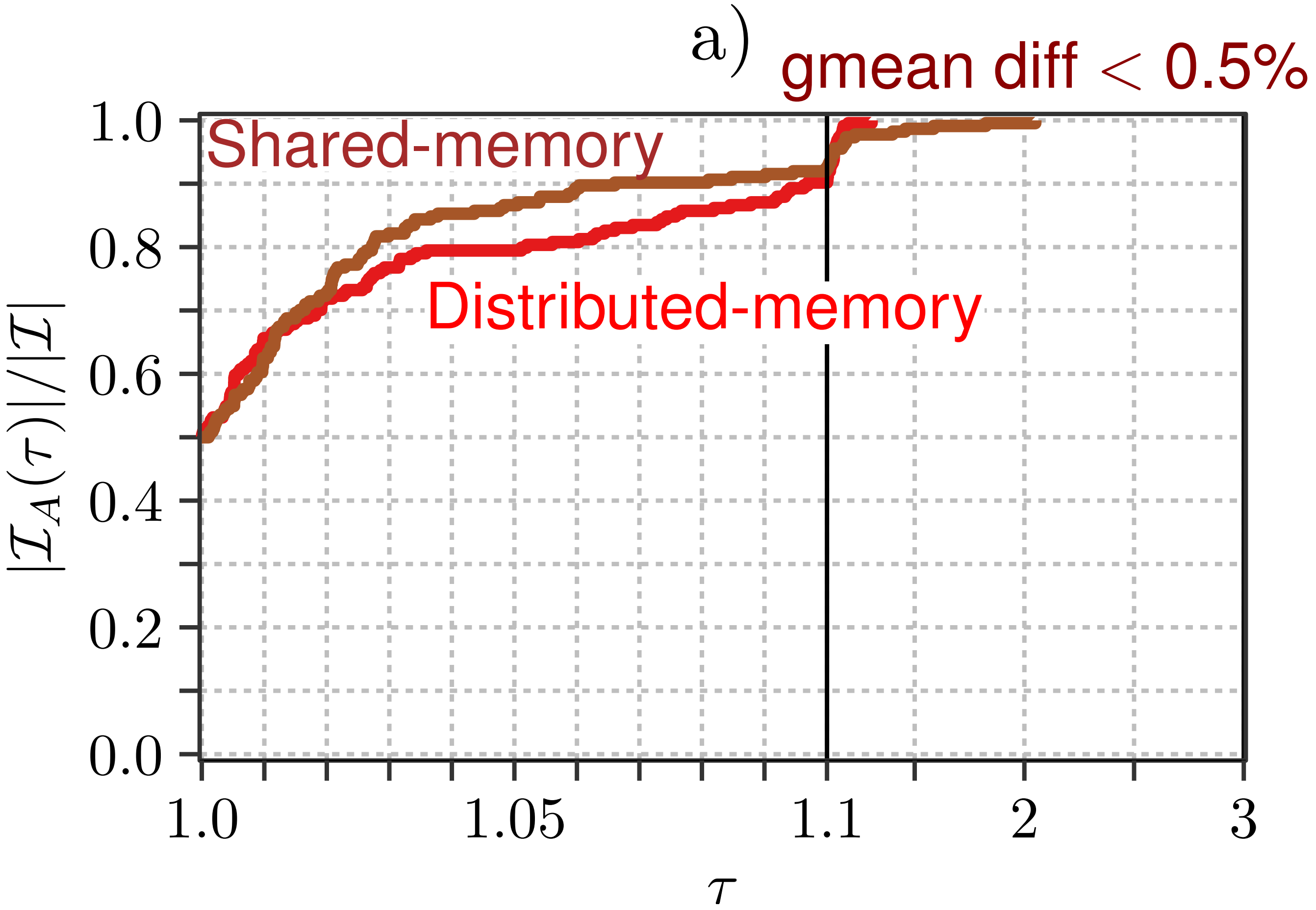
[1 node @ 64 cores]

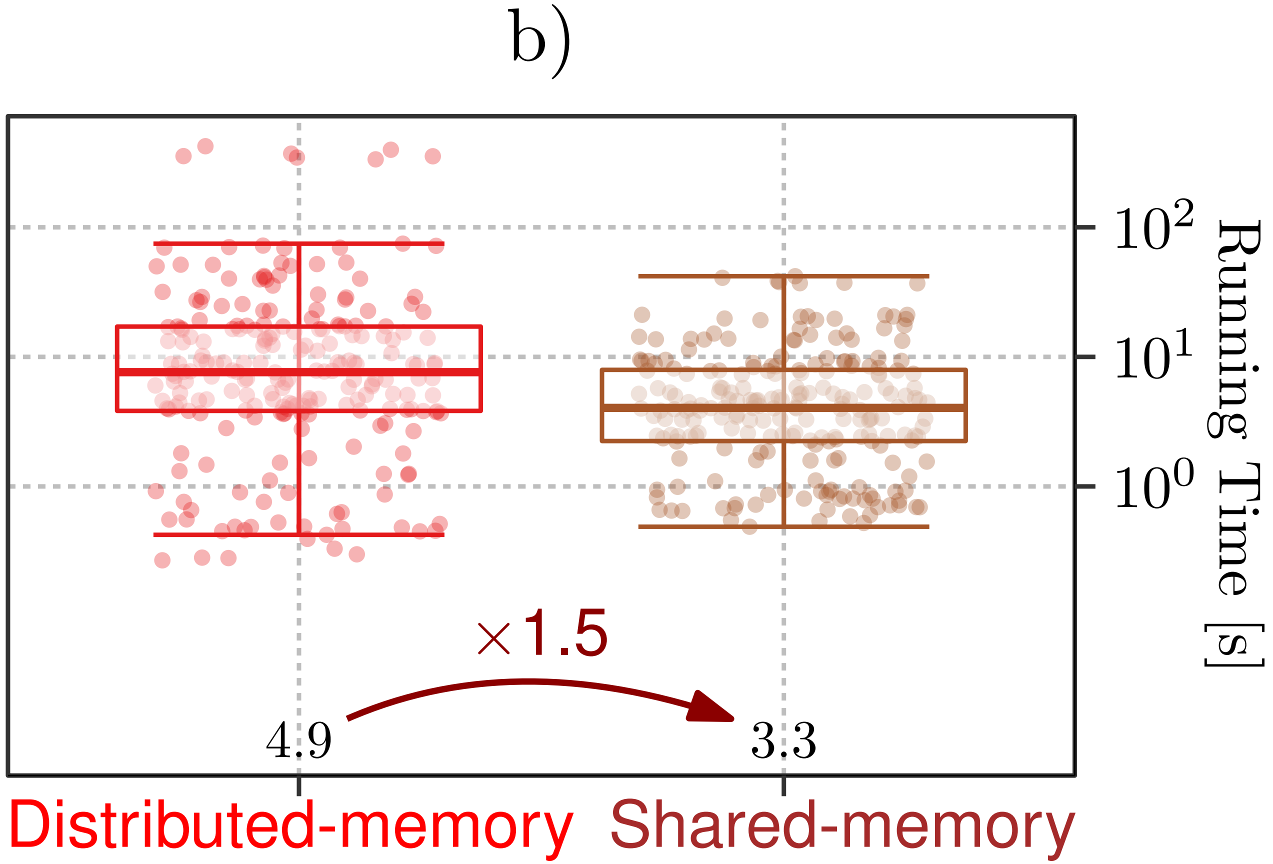
Experiments – Quality: vs. Shared-Memory



[1 node @ 64 cores]

Experiments – Quality: vs. Shared-Memory





[1 node @ 64 cores]

Conclusion

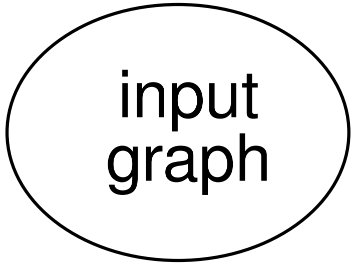
- Deep Multilevel Graph Partitioning:
  - Integrate coarsening deep into initial partitioning

- **dKaMinPar**: distributed deep MGP implementation
  - Scales to thousands of PEs, competitive partition quality
  - Better scalability for large *k* than previous approaches

- **Future**: stronger distributed refinement algorithms

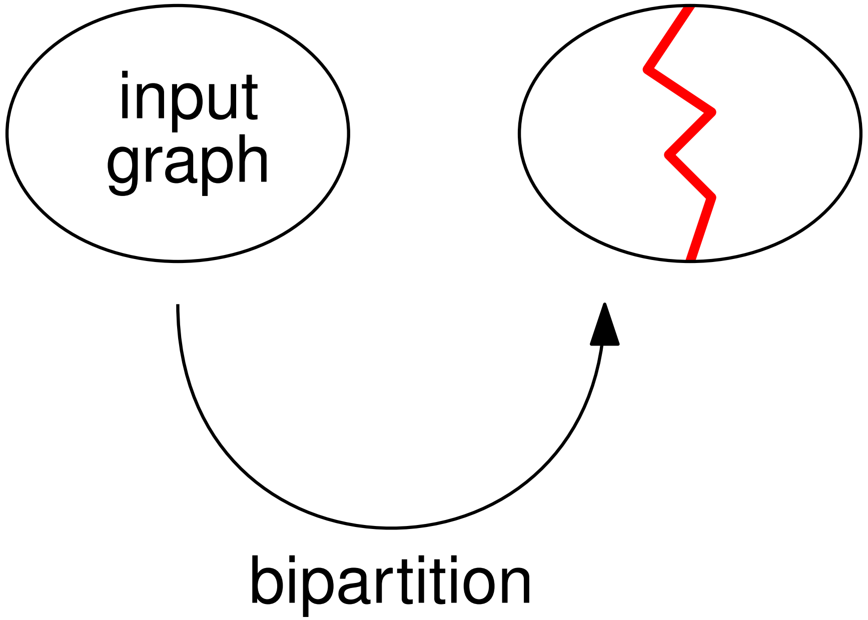
- Supplementary data available online:
  - Full experimental results: `algo2.iti.kit.edu/seemaier/ddeep_mgp/`
  - Source code: `github.com/KaHIP/KaMinPar`

MGP: Recursive Bipartitioning

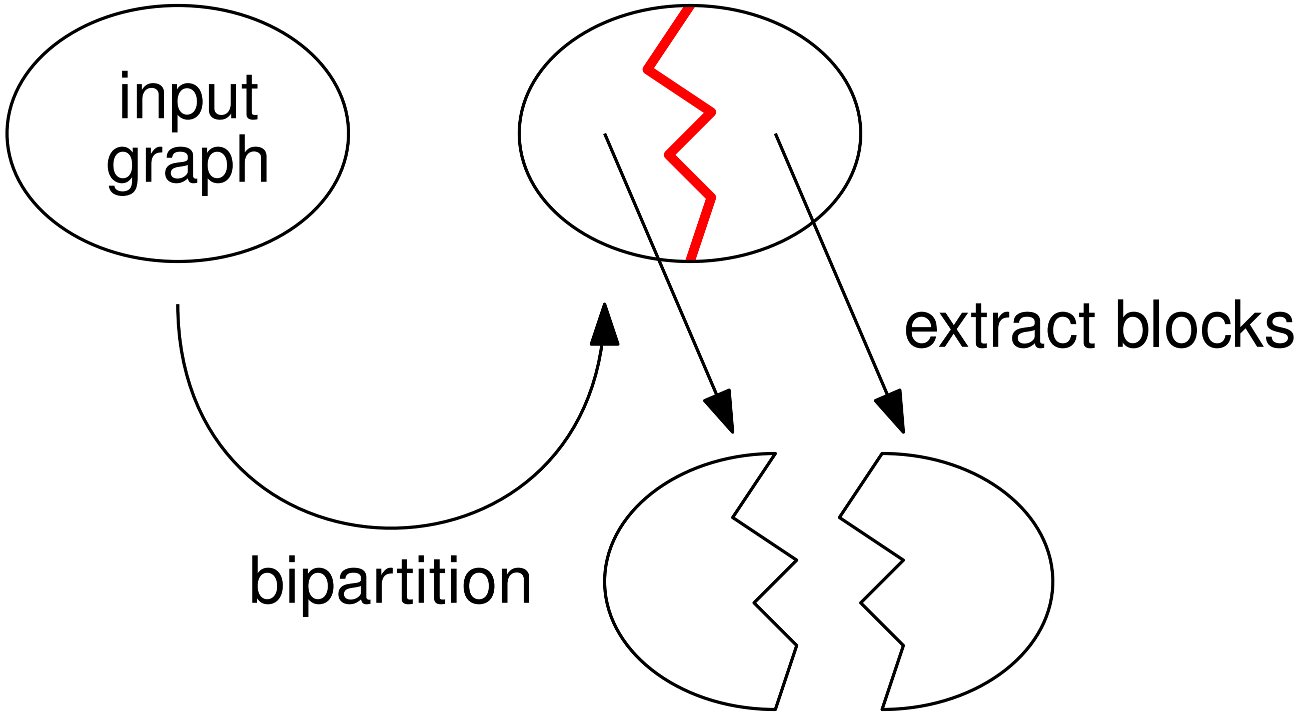


input graph

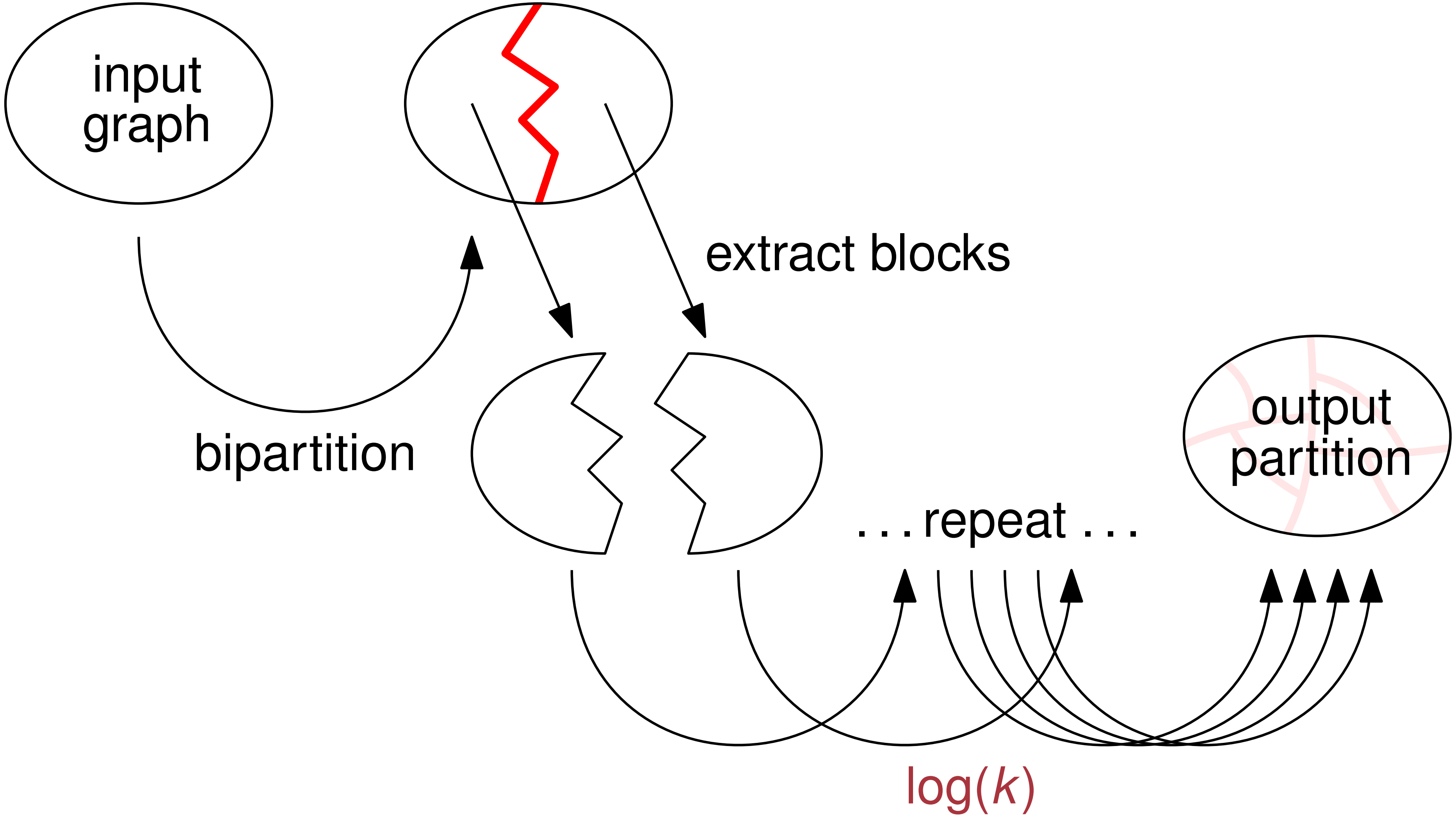
MGP: Recursive Bipartitioning



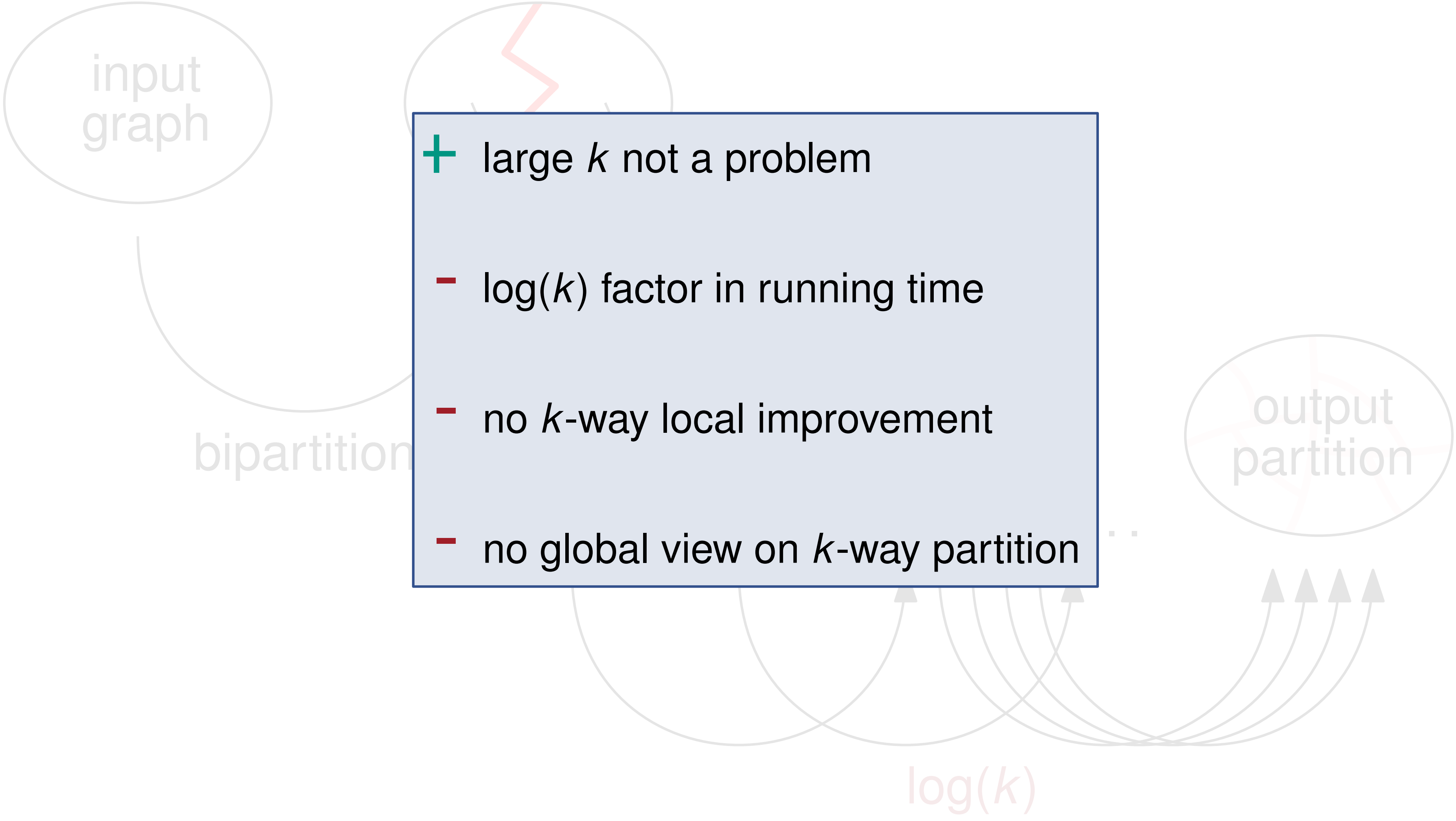
MGP: Recursive Bipartitioning



MGP: Recursive Bipartitioning



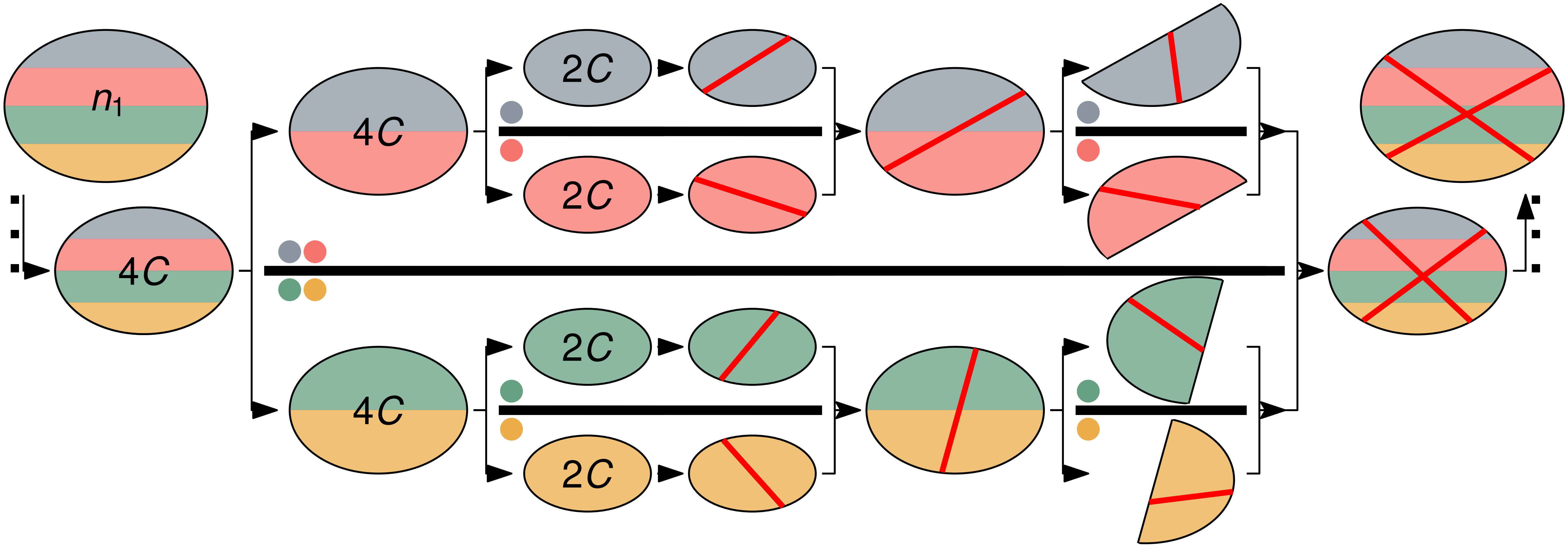
MGP: Recursive Bipartitioning



- + large *k* not a problem
- - log(*k*) factor in running time
- - no *k*-way local improvement
- - no global view on *k*-way partition

dKaMinPar – Refinement

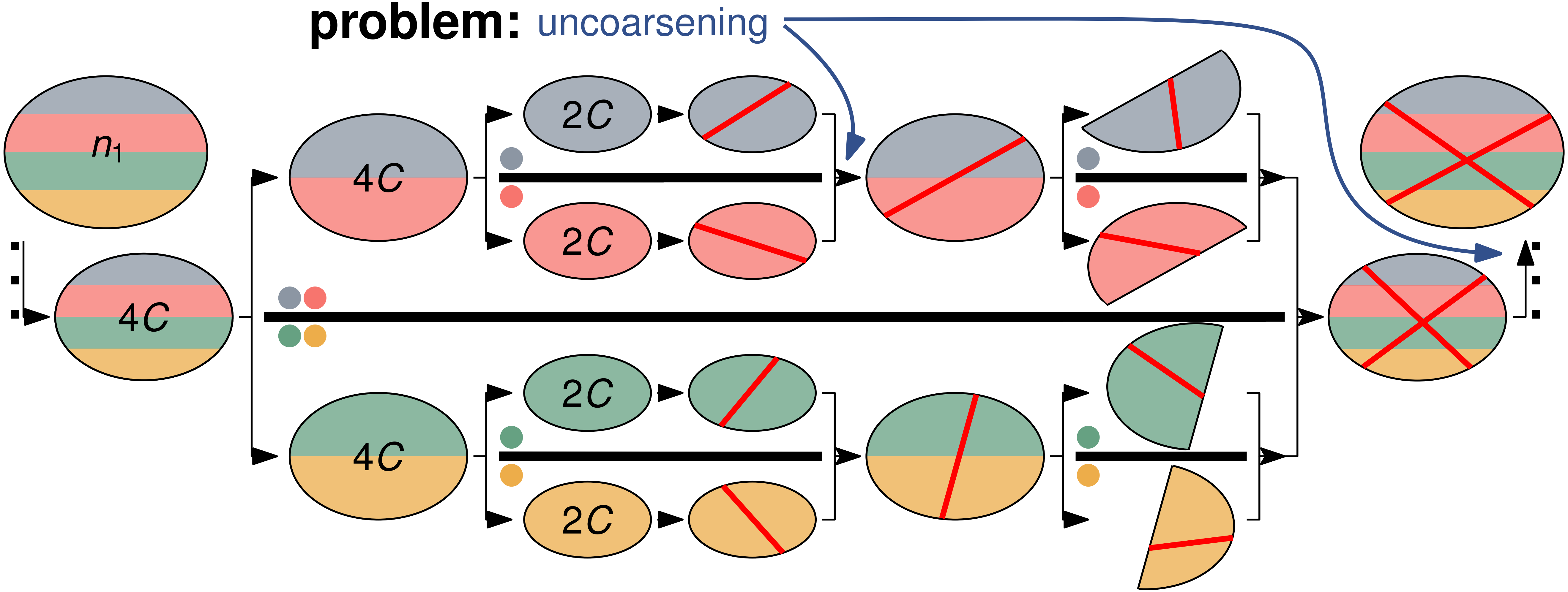
recall: $c(V_i) \leq \max\left\{(1+\varepsilon)\frac{c(V)}{k}, \frac{c(V)}{k} + \max_v c(v)\right\}$



dKaMinPar – Refinement

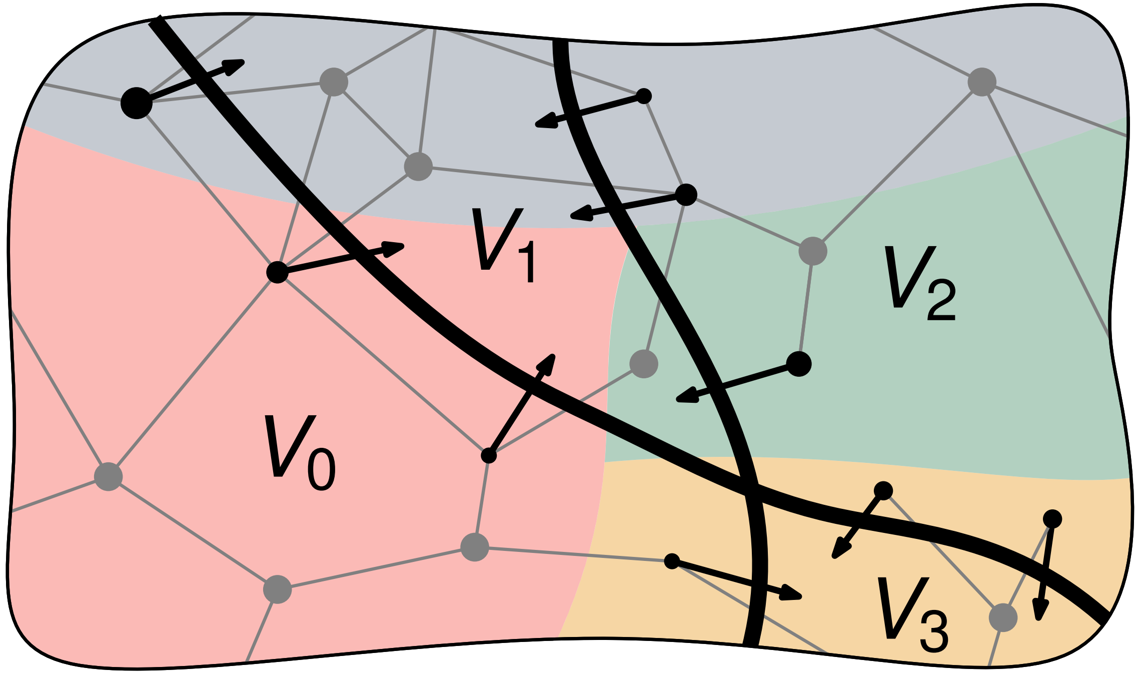
recall: $c(V_i) \leq \max\left\{(1+\varepsilon)\frac{c(V)}{k}, \frac{c(V)}{k} + \max_v c(v)\right\}$

problem: uncoarsening



dKaMinPar – Refinement

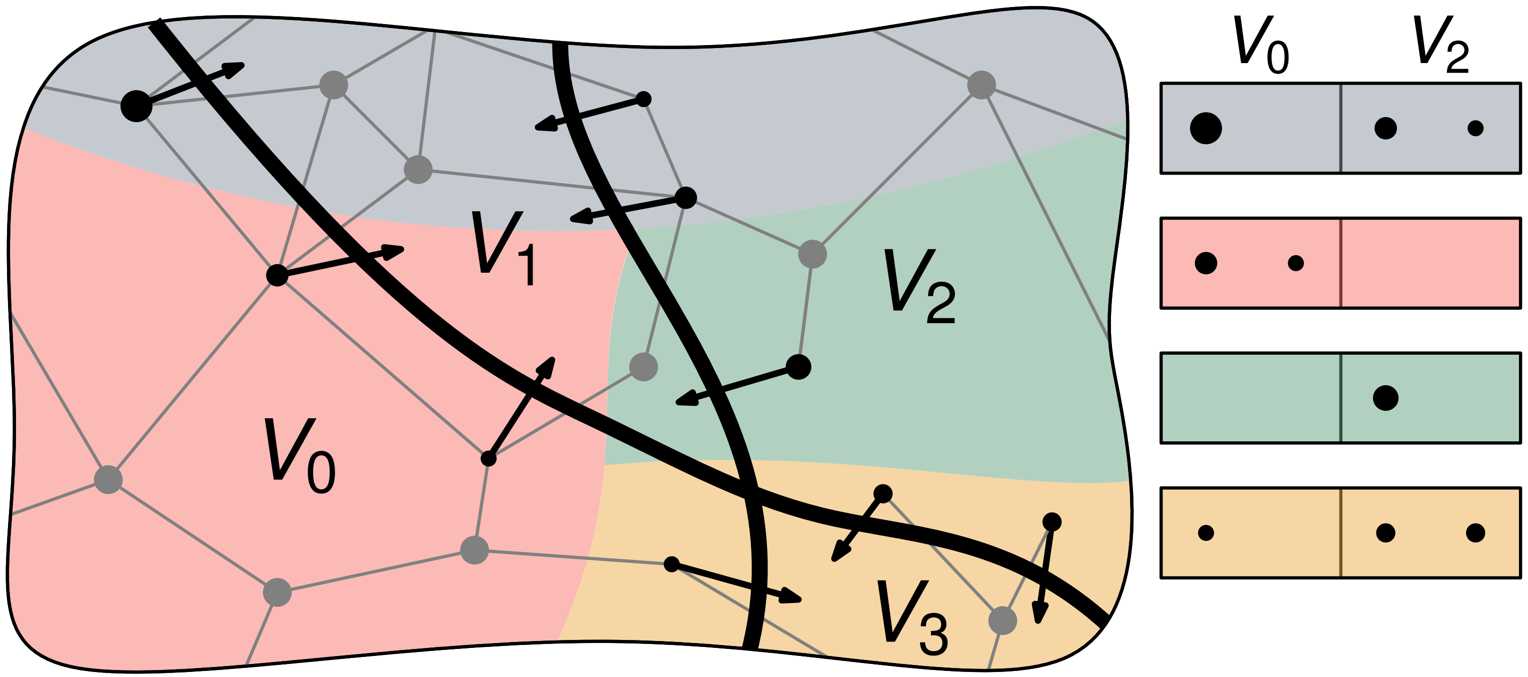
- Refinement: label propagation + balancing



●●●● PEs

dKaMinPar – Refinement

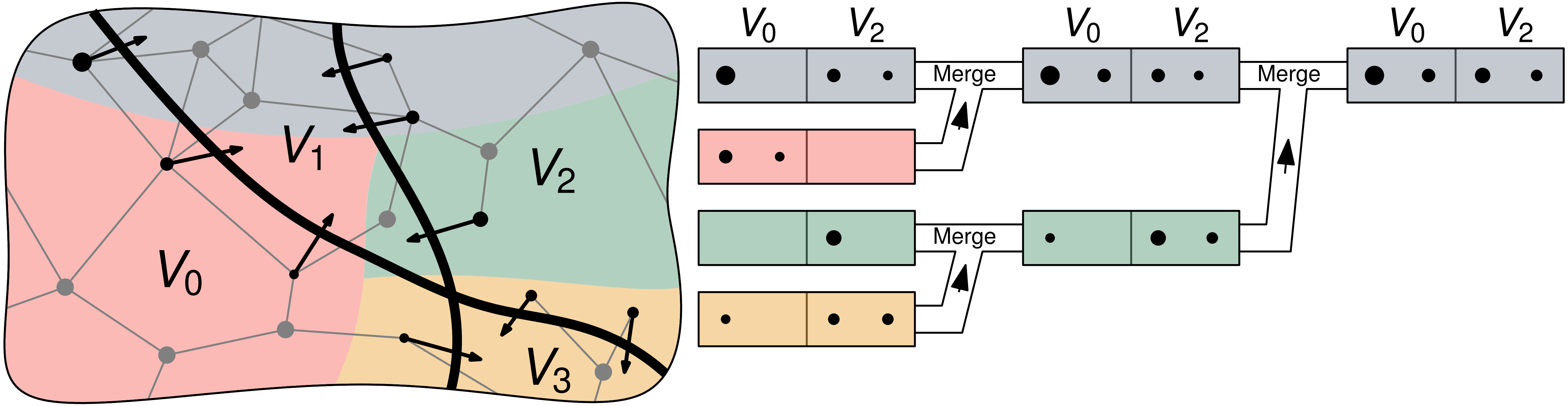
- Refinement: label propagation + balancing



●●●● PEs

dKaMinPar – Refinement

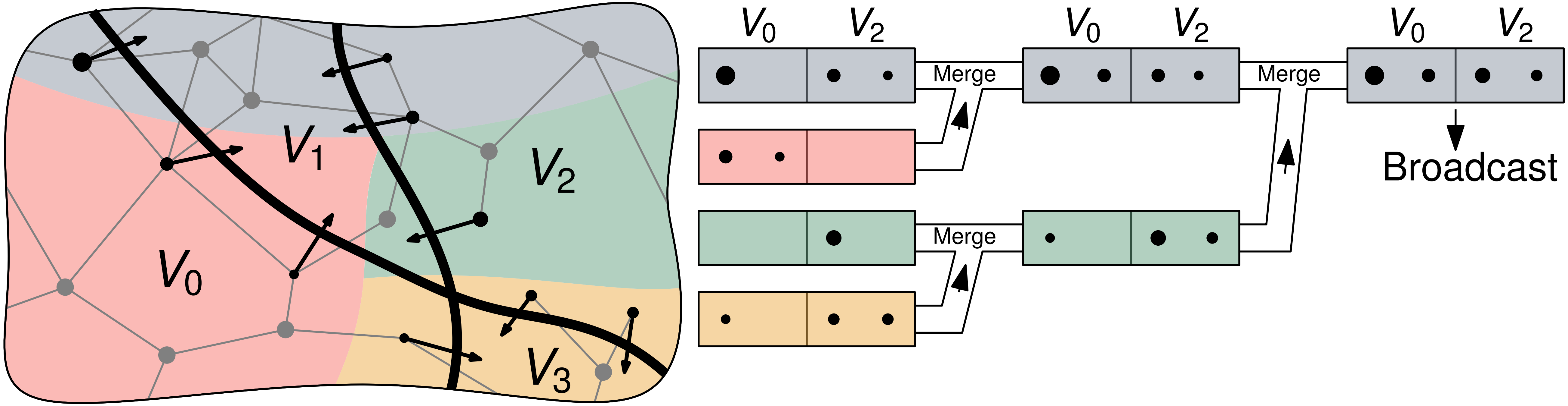
- Refinement: label propagation + balancing



PEs

dKaMinPar – Refinement

- Refinement: label propagation + balancing



●●●● PEs