Deep Multilevel Graph Partitioning

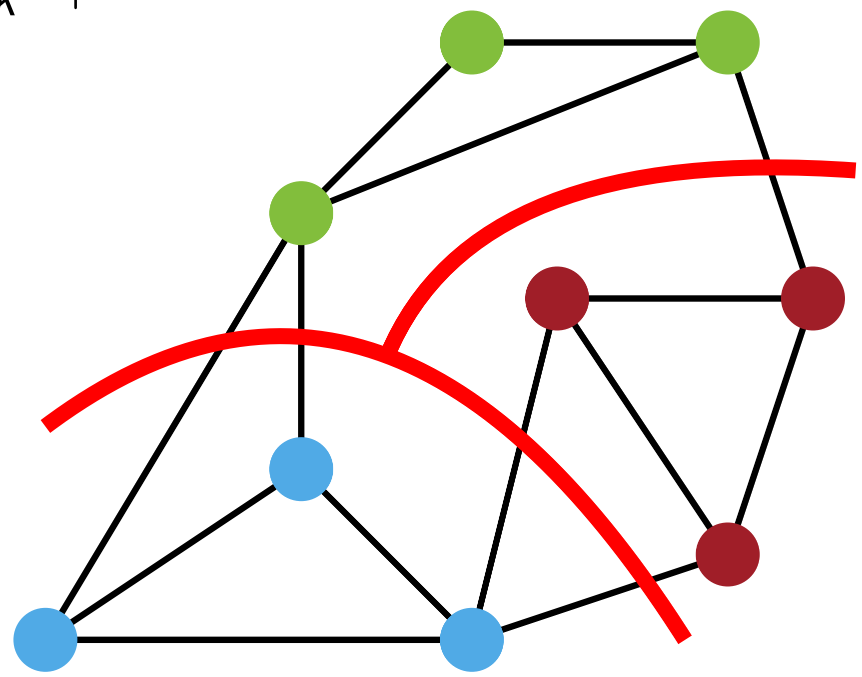
September 7, 2021

Lars Gottesbüren, Tobias Heuer, Peter Sanders, Christian Schulz, 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: $c(V_i) \leq (1 + \varepsilon)\lceil \frac{c(V)}{k} \rceil$
- while minimizing the edge cut: $\sum_{i \neq j} \omega(E_{ij})$

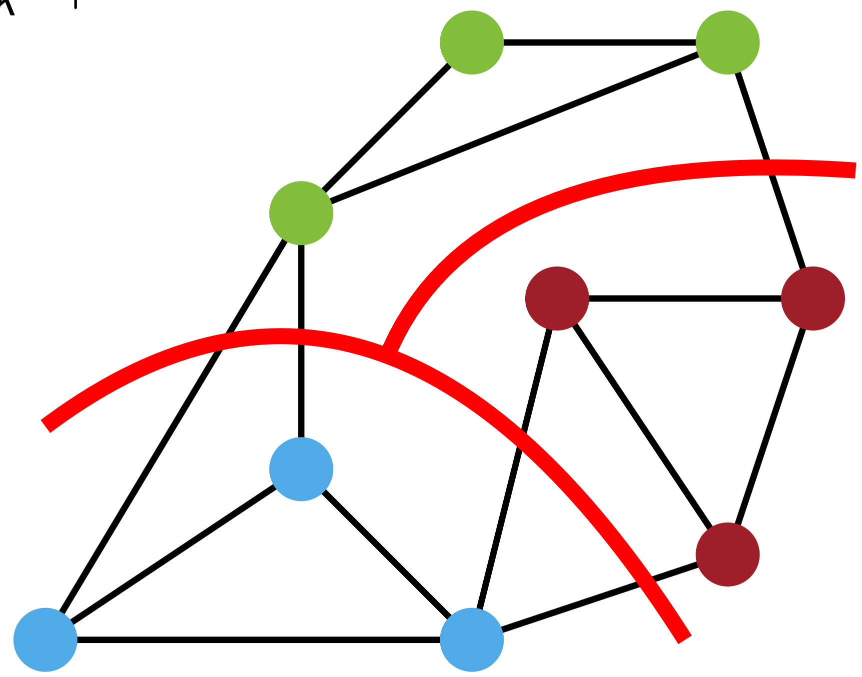


Graph Partitioning

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

node weights → c ; edge weights → ω

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



Graph Partitioning

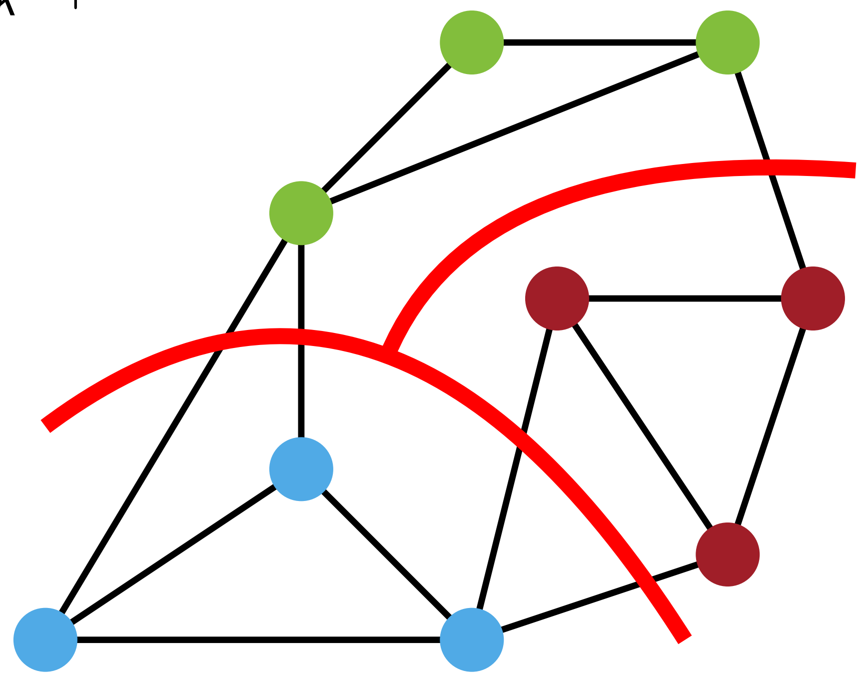
Given a graph $G = (V, E, c, \omega)$, partition V into k disjoint blocks such that:

node weights (→ c) edge weights (→ ω)

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

imbalance factor (→ ε)

- 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:

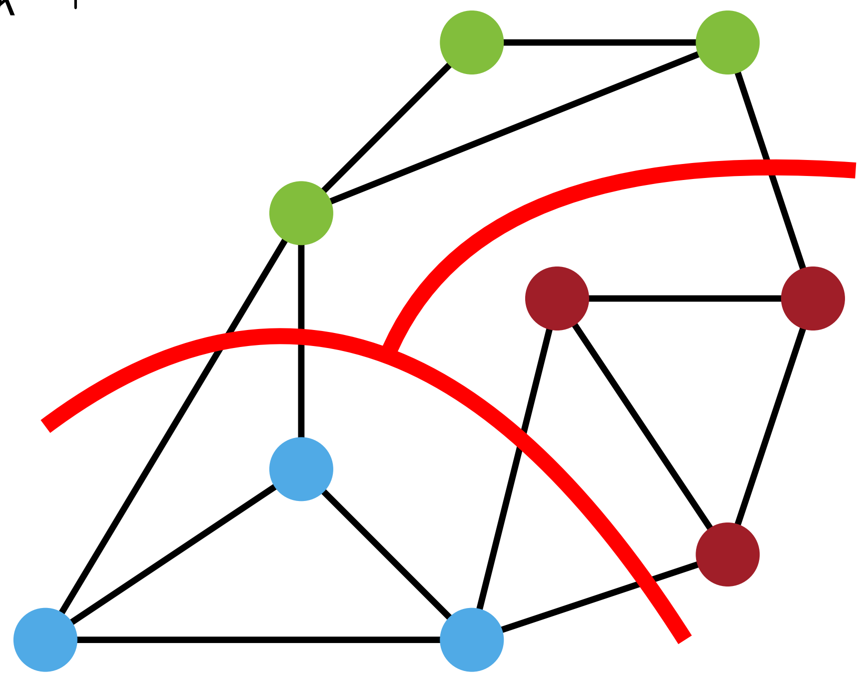
node weights → c; edge weights → ω

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

  imbalance factor → ε

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

  edges between blocks i and j → E_{ij}



Graph Partitioning

Karlsruhe Institute of Technology

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

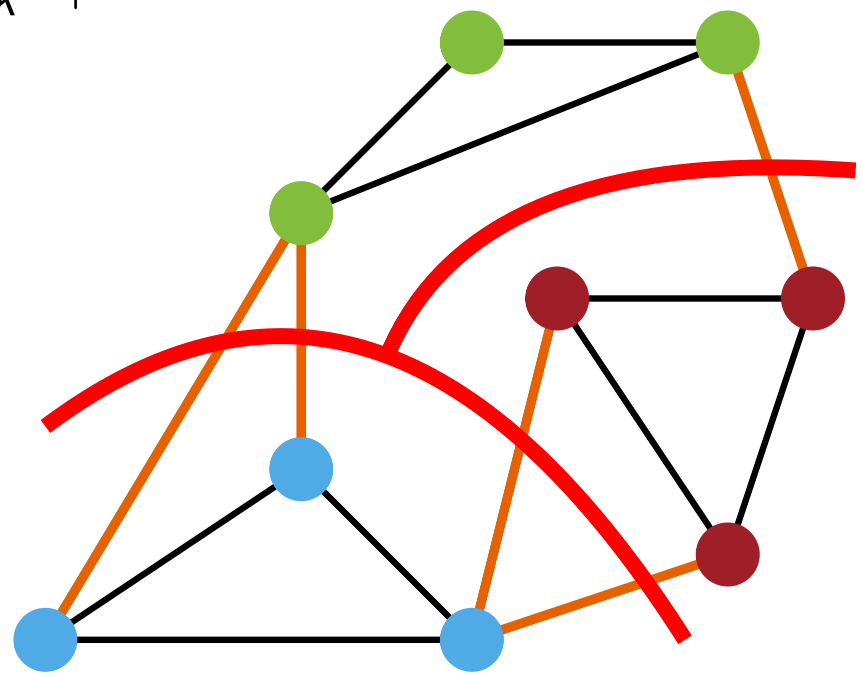
node weights (→ c) edge weights (→ ω)

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

  imbalance factor (→ ε)

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

  edges between blocks i and j (→ E_{ij})



Graph Partitioning for Parallel Computing

- Distributed graph across PEs
  minimize communication between PEs

[HoreKa, KIT]

- Available parallelism increases steadily

- Established GPs tools are not designed to handle large k

Graph Partitioning for Parallel Computing

- Distributed graph across PEs
  minimize communication between PEs

- Available parallelism increases steadily

[HoreKa, KIT]

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

our contribution: improve state-of-the-art there

Graph Partitioning for Parallel Computing

- Distributed graph across PEs
  minimize communication between PEs

[HoreKa, KIT]

- Available parallelism i

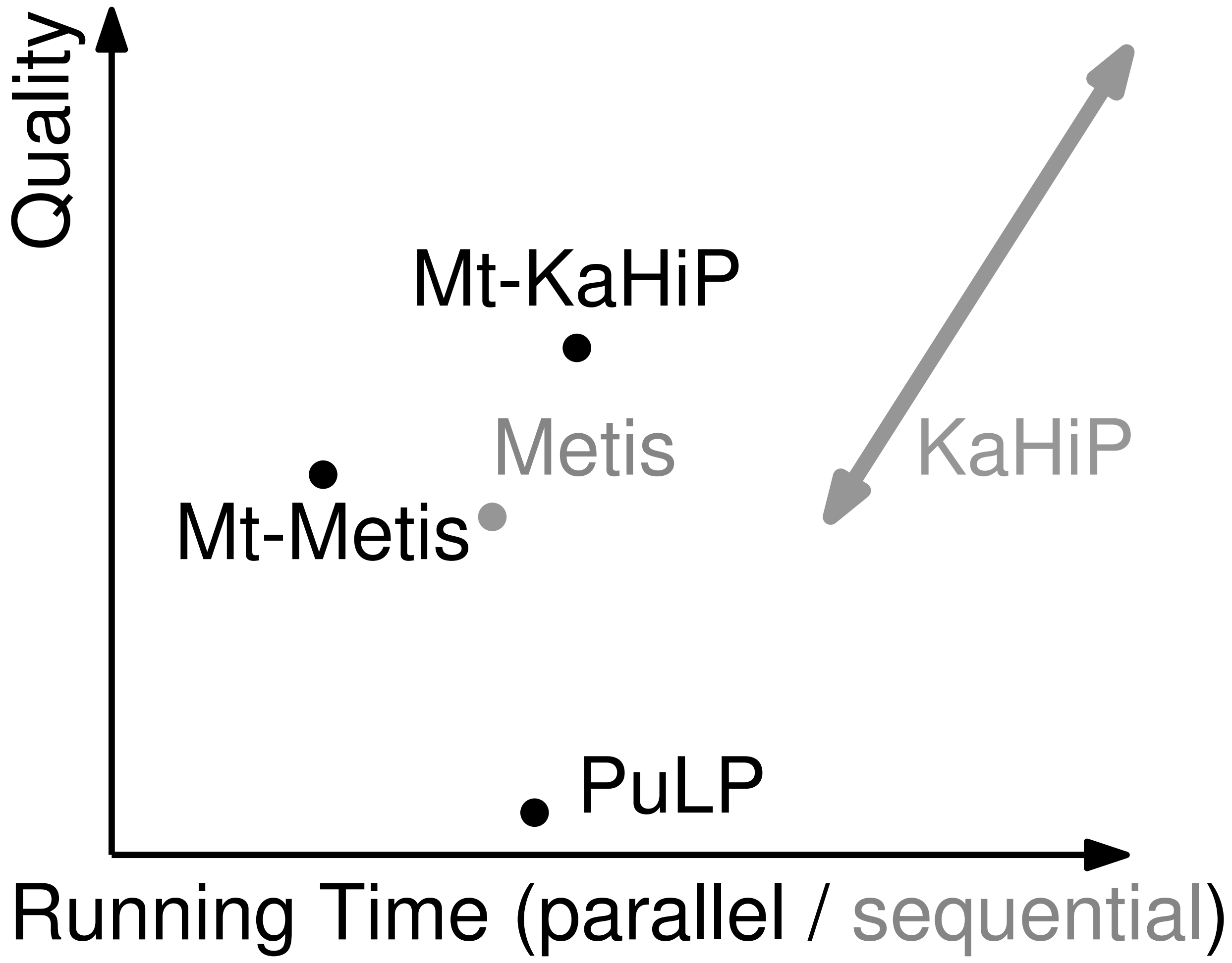
> Graph partitioning is NP-complete
> $\Rightarrow$ we focus on heuristics

- Established GPs tools are not designed to handle large k

our contribution: improve state-of-the-art there

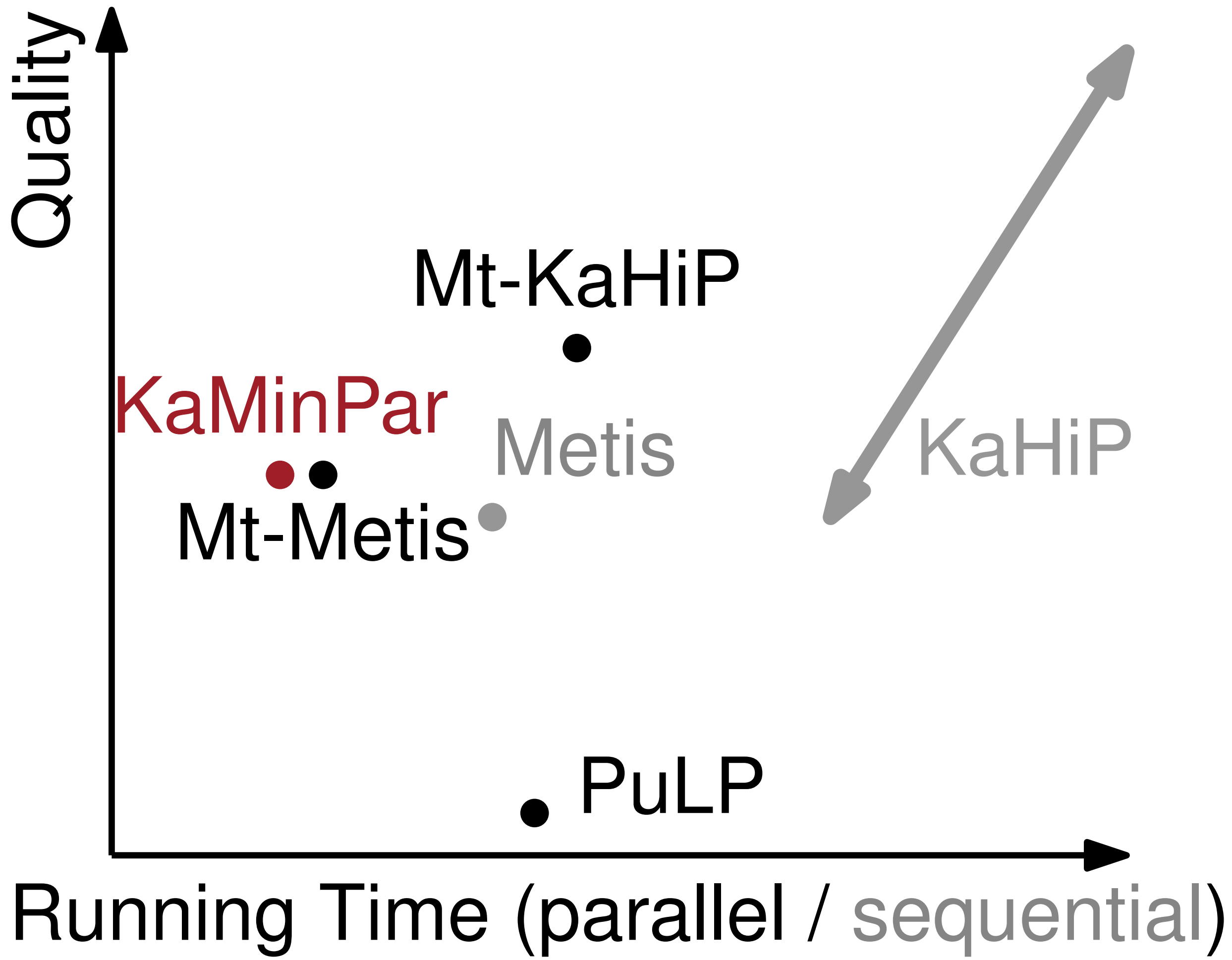
Graph Partitioning Tools: our Contribution

"Small" k

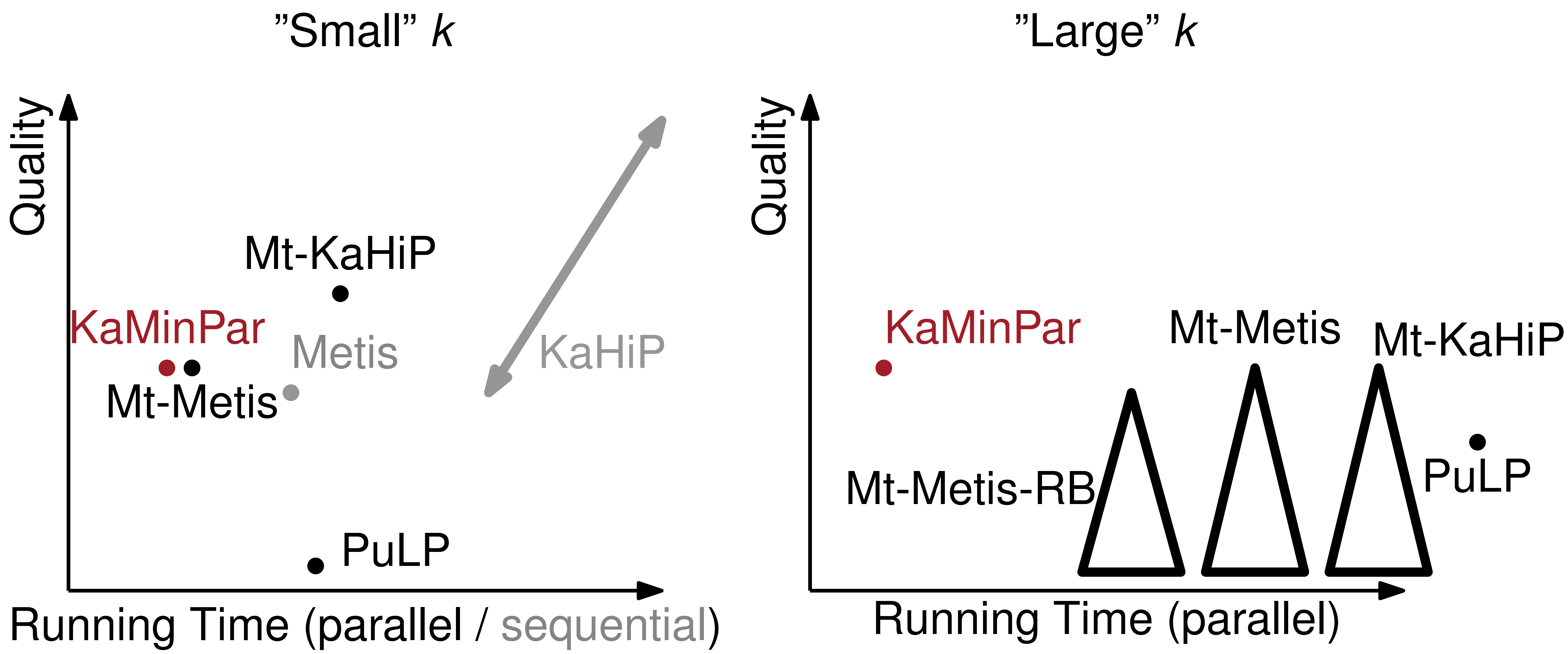


Graph Partitioning Tools: our Contribution

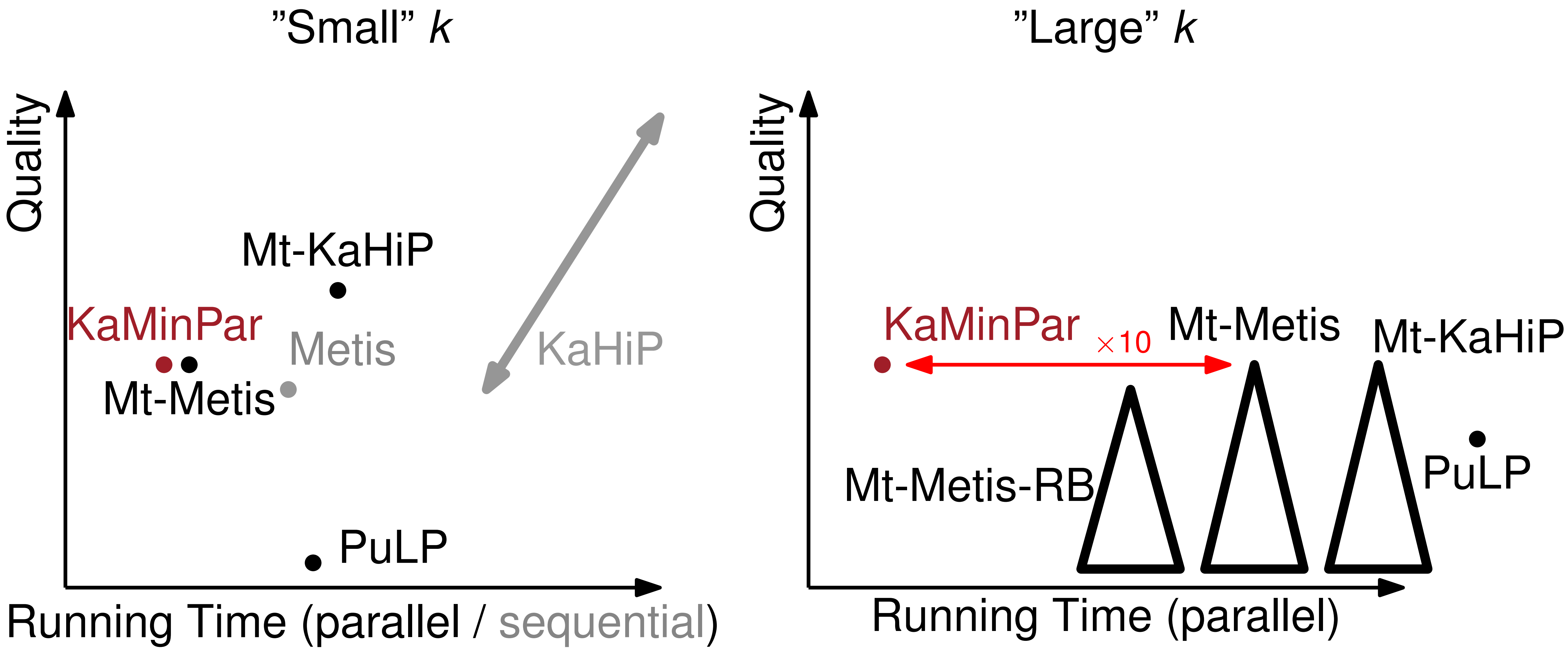
"Small" k



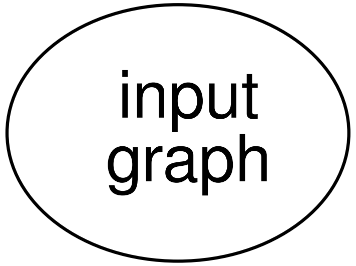
Graph Partitioning Tools: our Contribution



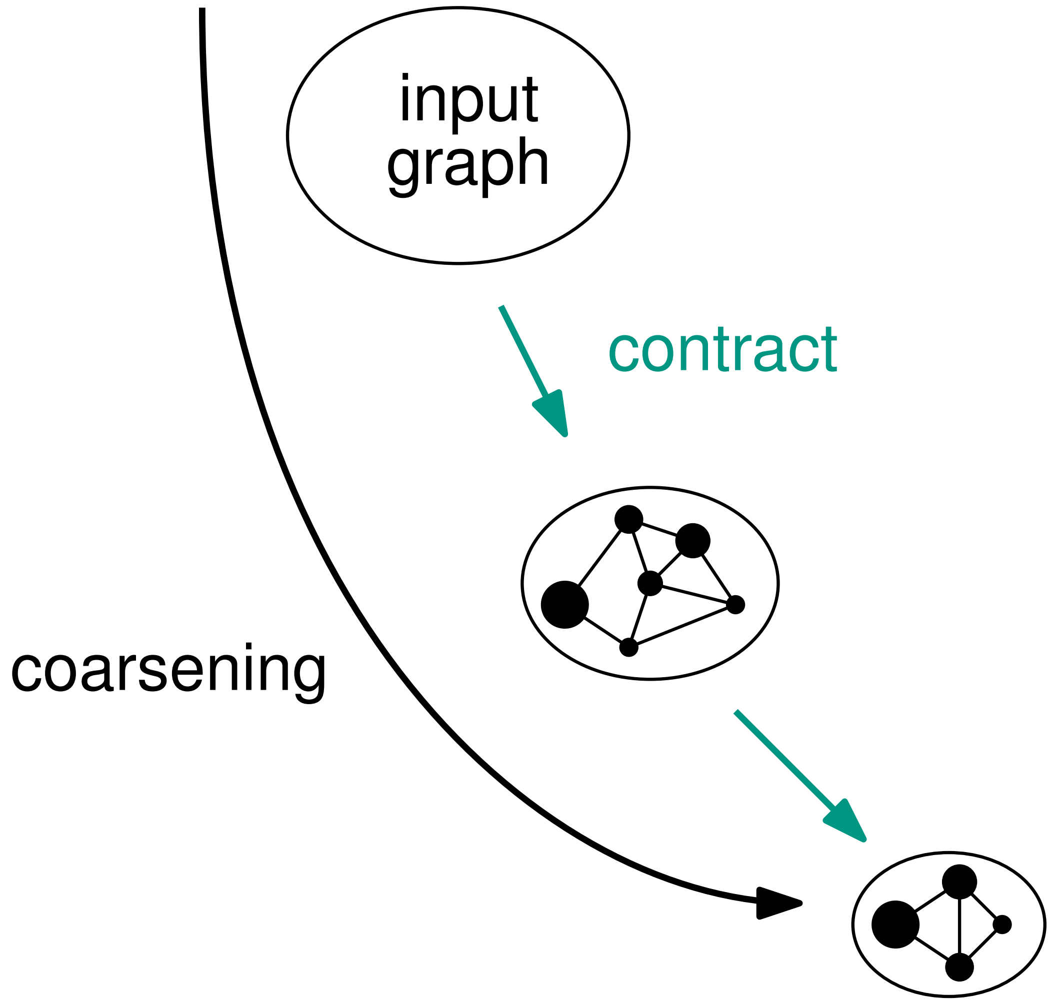
Graph Partitioning Tools: our Contribution



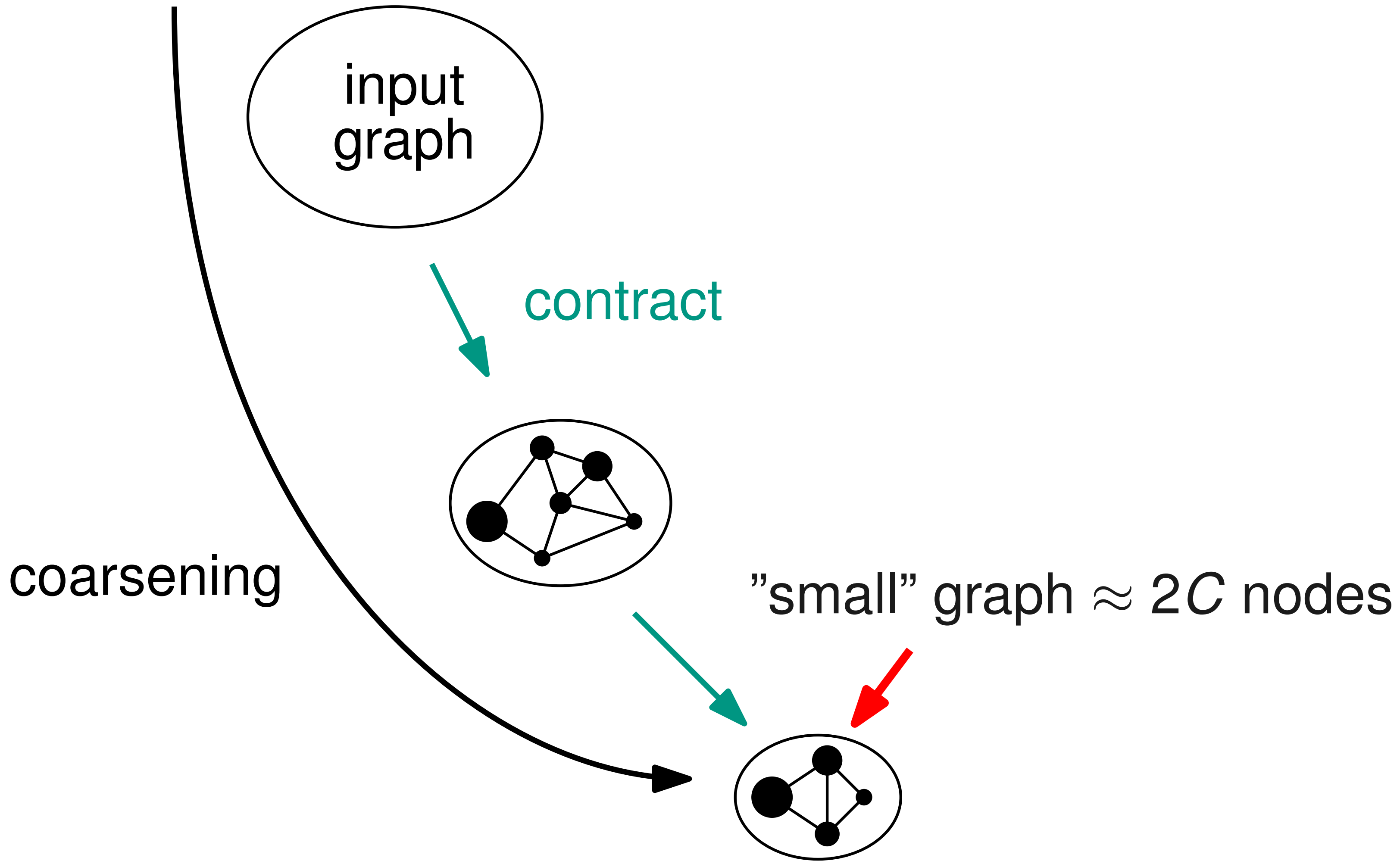
Multilevel Graph Partitioning



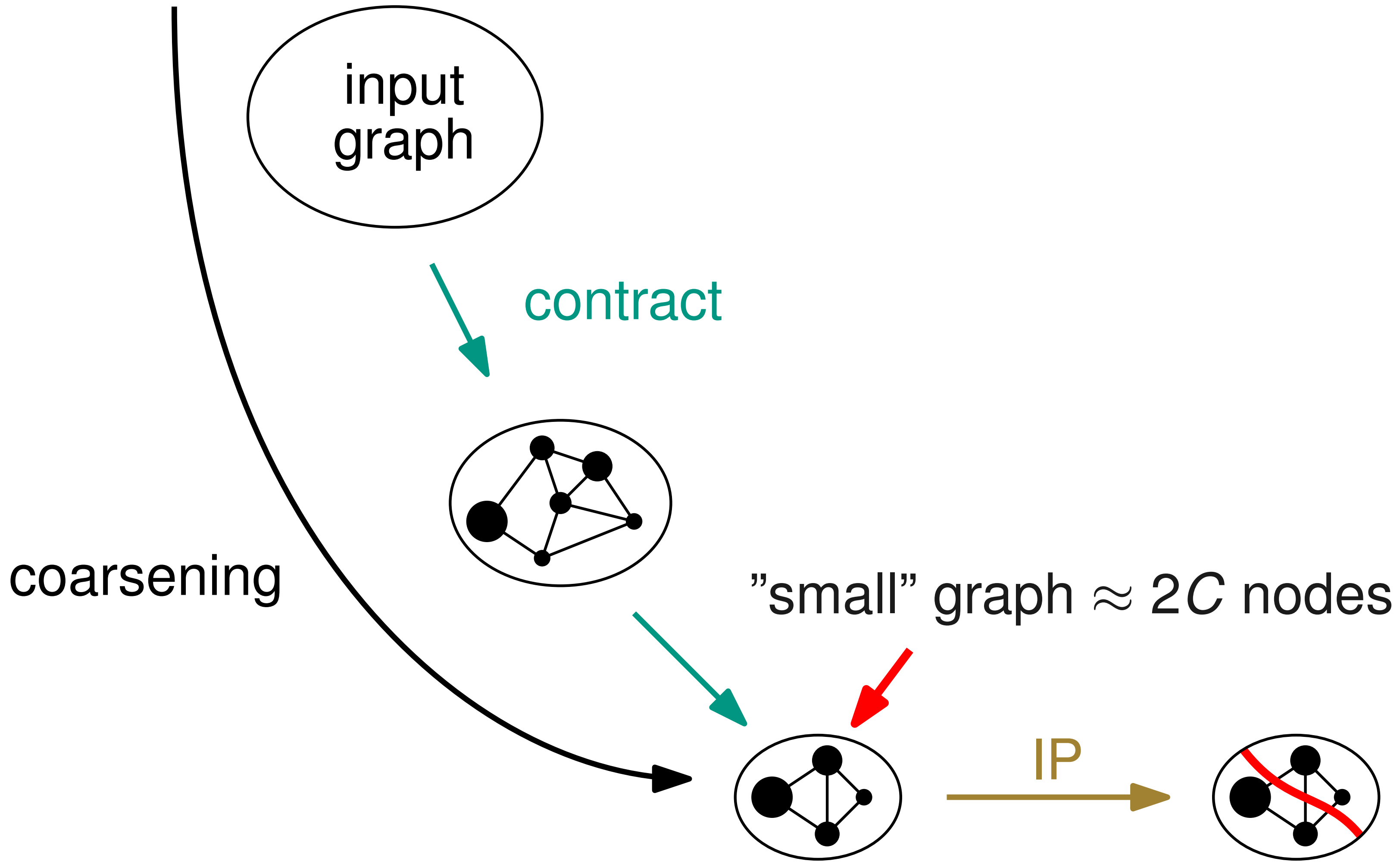
Multilevel Graph Partitioning



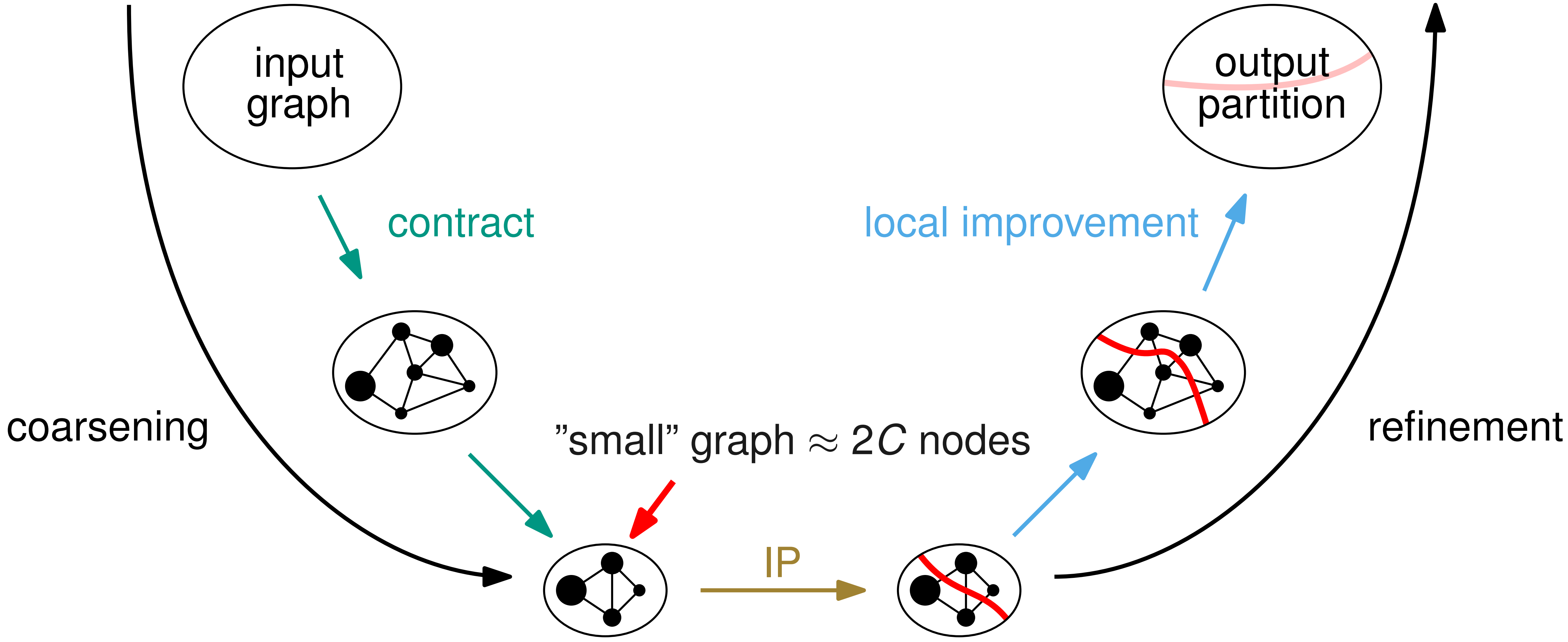
Multilevel Graph Partitioning



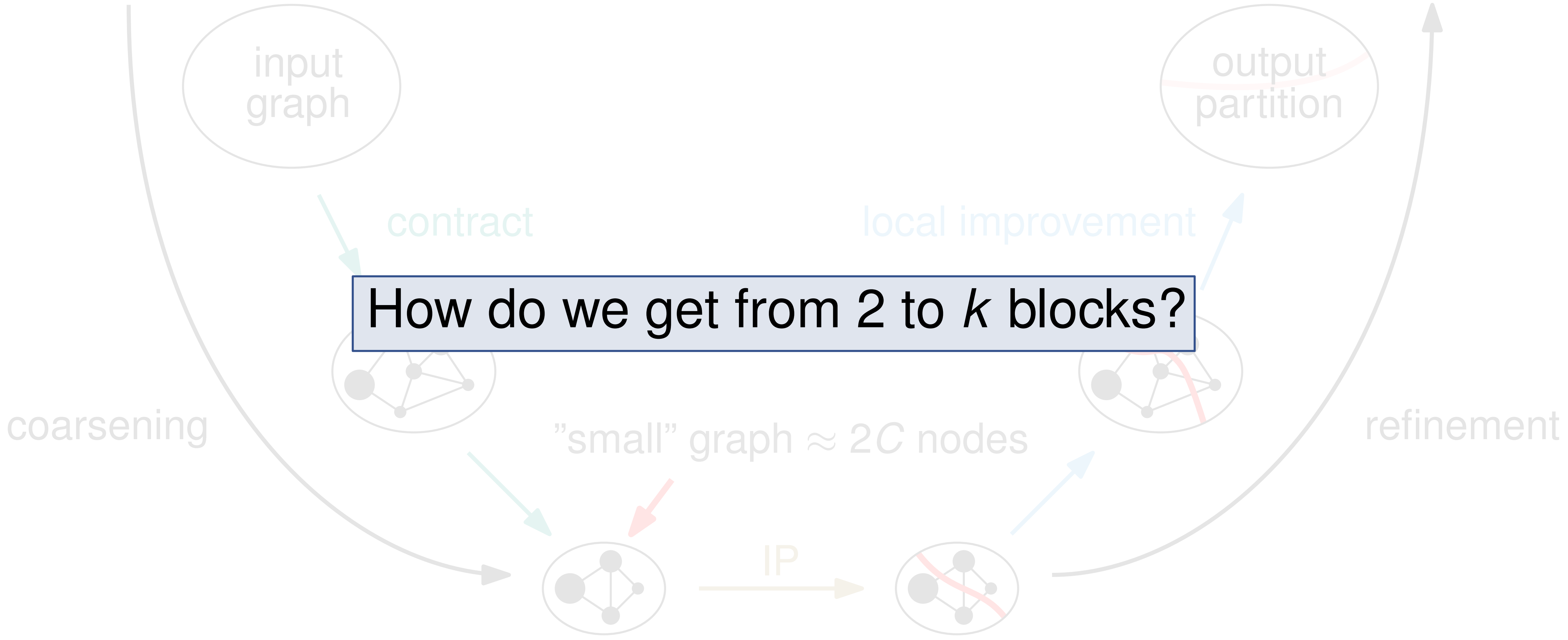
Multilevel Graph Partitioning



Multilevel Graph Partitioning

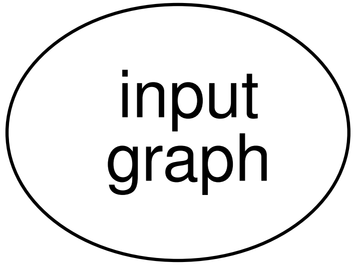


Multilevel Graph Partitioning

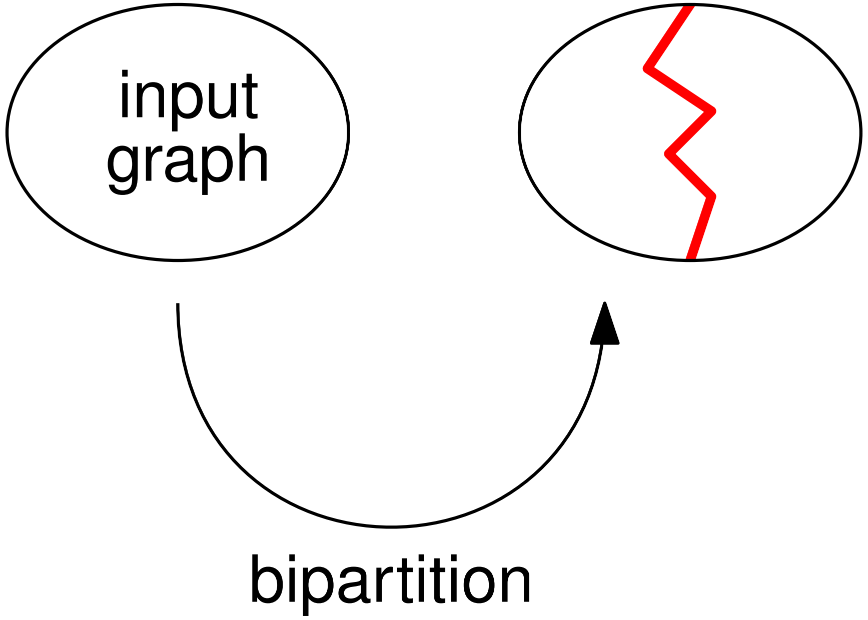


How do we get from 2 to k blocks?

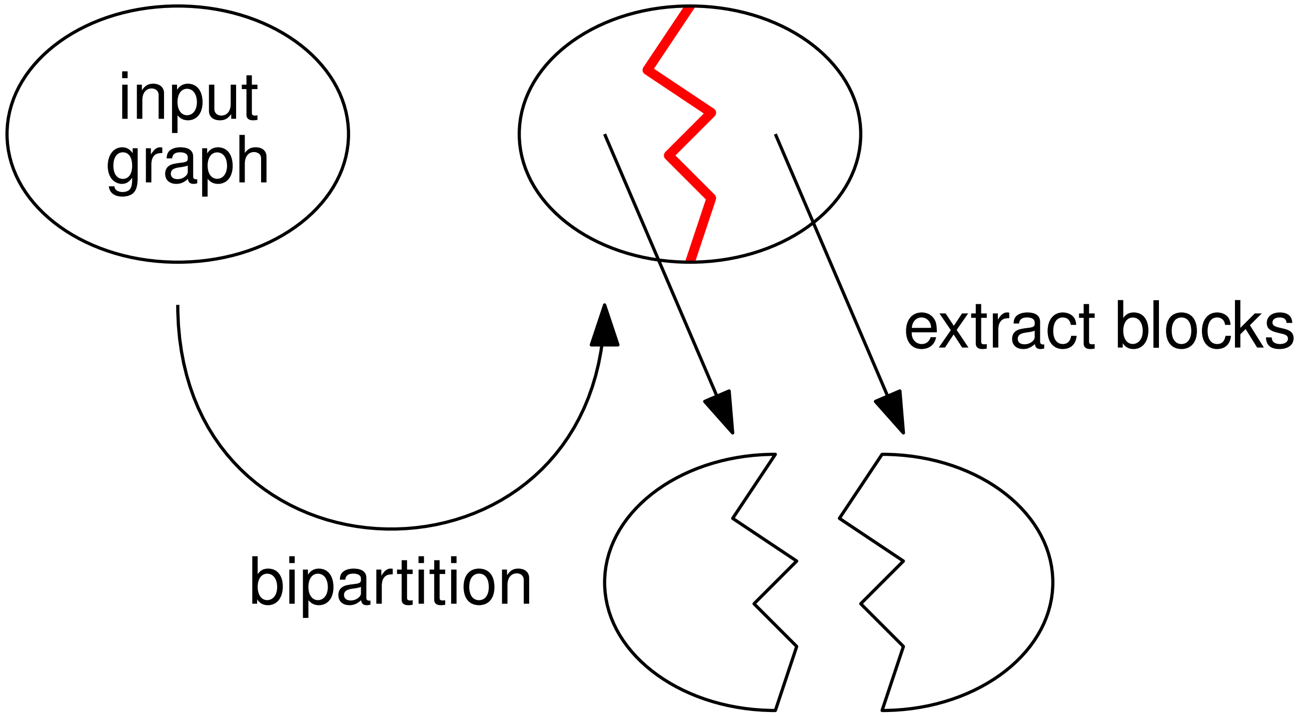
MGP: Recursive Bipartitioning



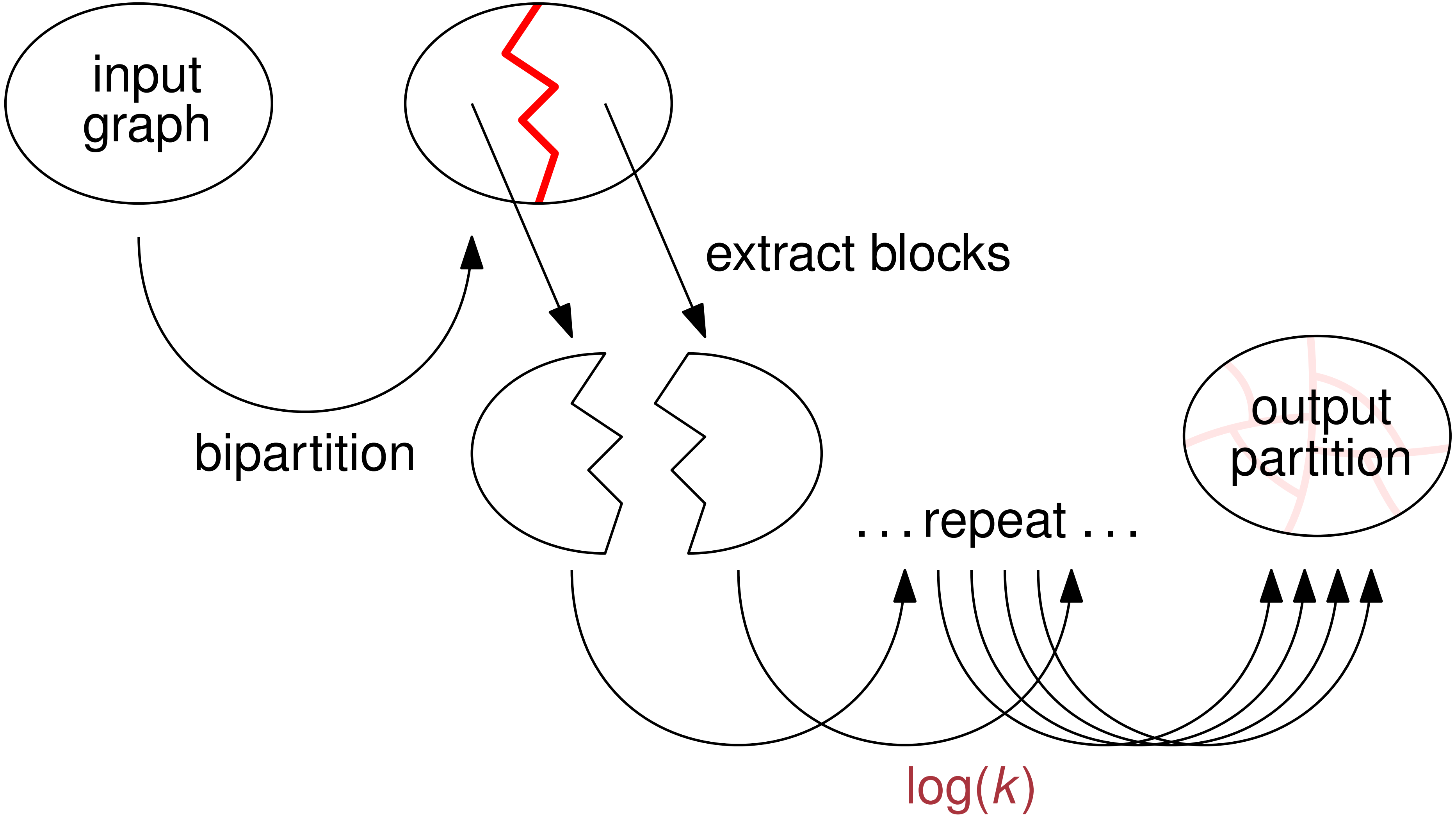
MGP: Recursive Bipartitioning



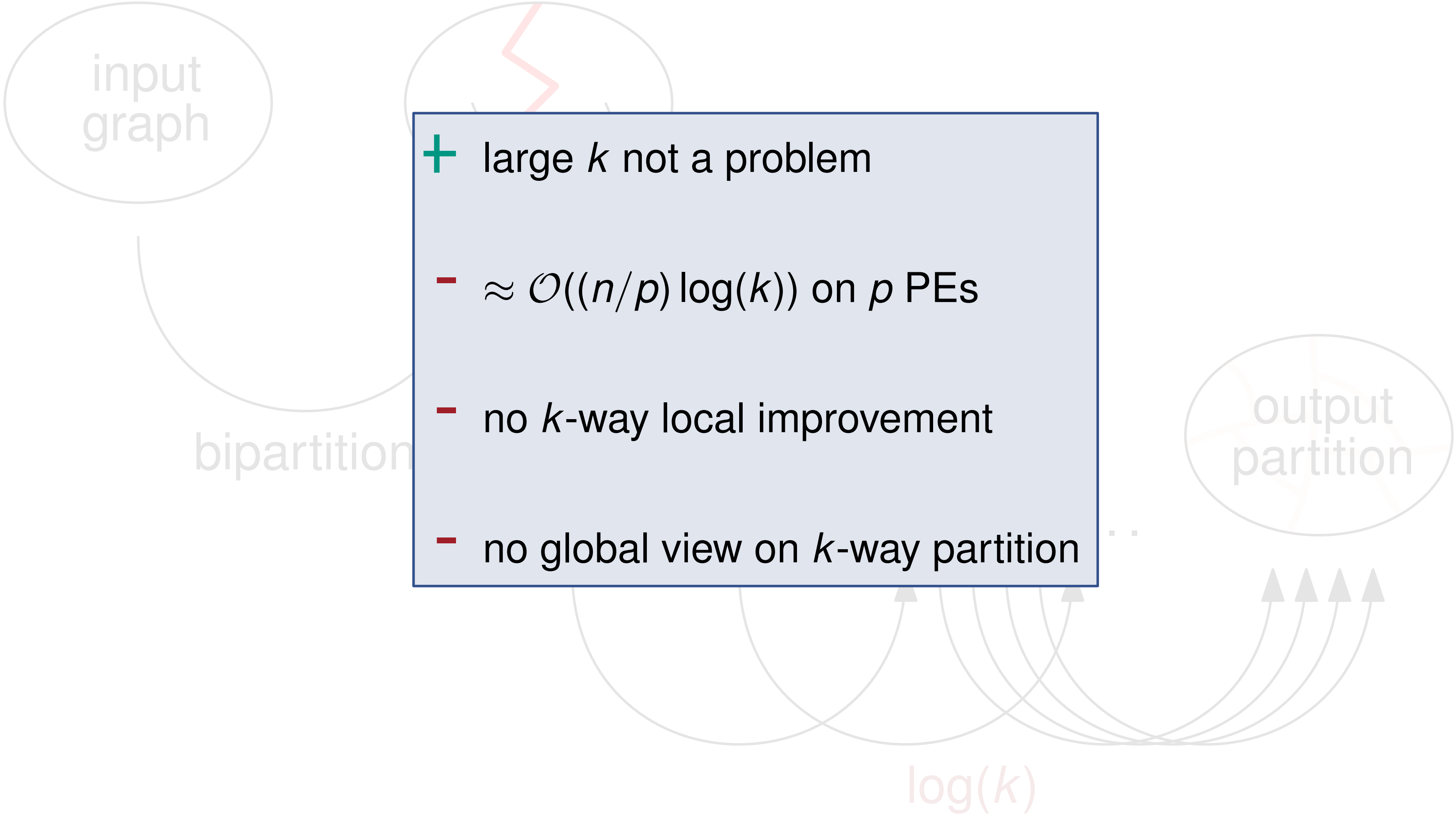
MGP: Recursive Bipartitioning



MGP: Recursive Bipartitioning

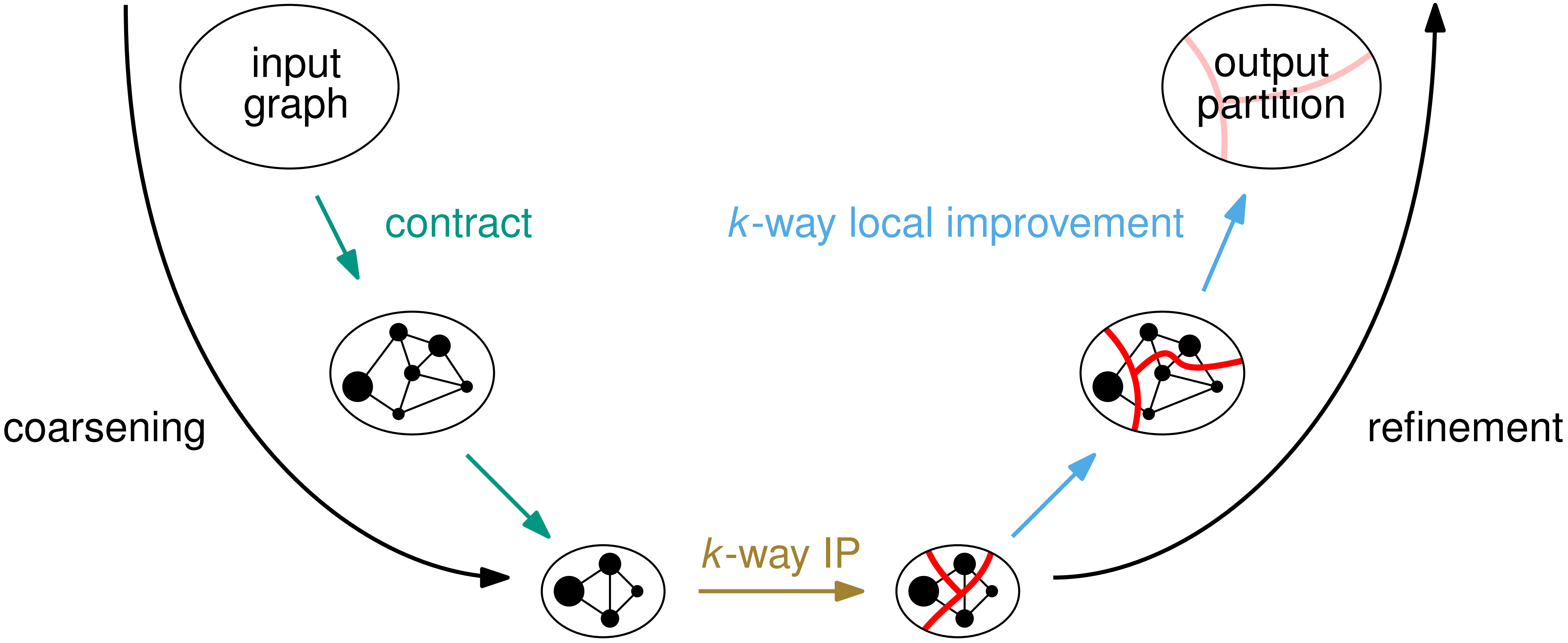


MGP: Recursive Bipartitioning

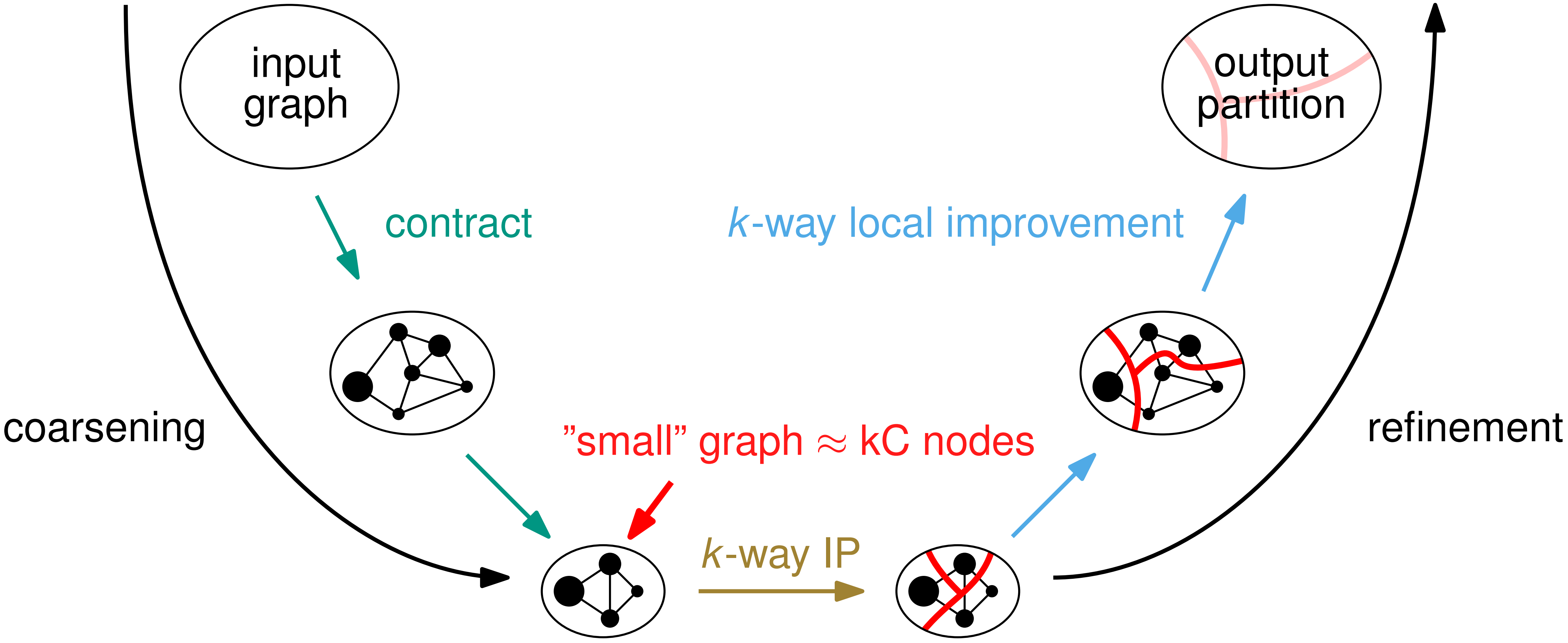


- \+ large k not a problem
- – $\approx \mathcal{O}((n/p)\log(k))$ on p PEs
- – no k-way local improvement
- – no global view on k-way partition

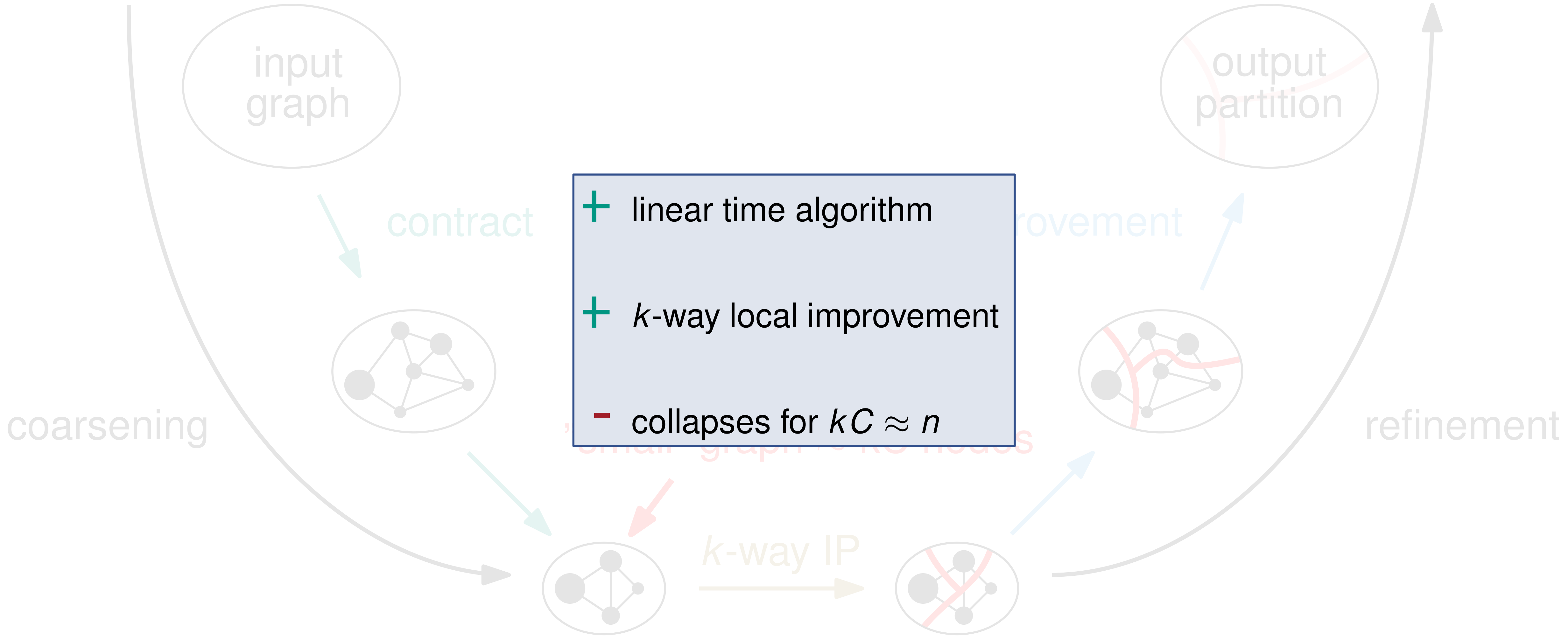
MGP: Direct *k*-way



MGP: Direct *k*-way



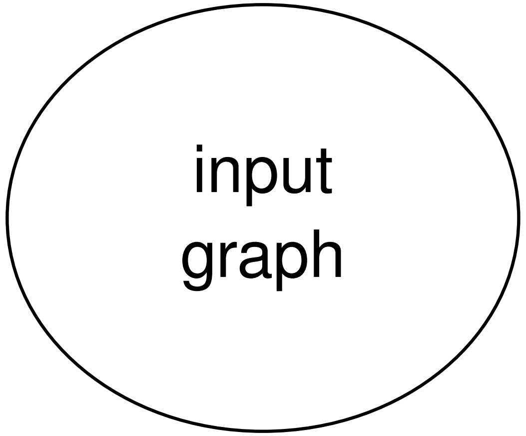
MGP: Direct k-way



- \+ linear time algorithm
- \+ k-way local improvement
- \- collapses for $kC \approx n$

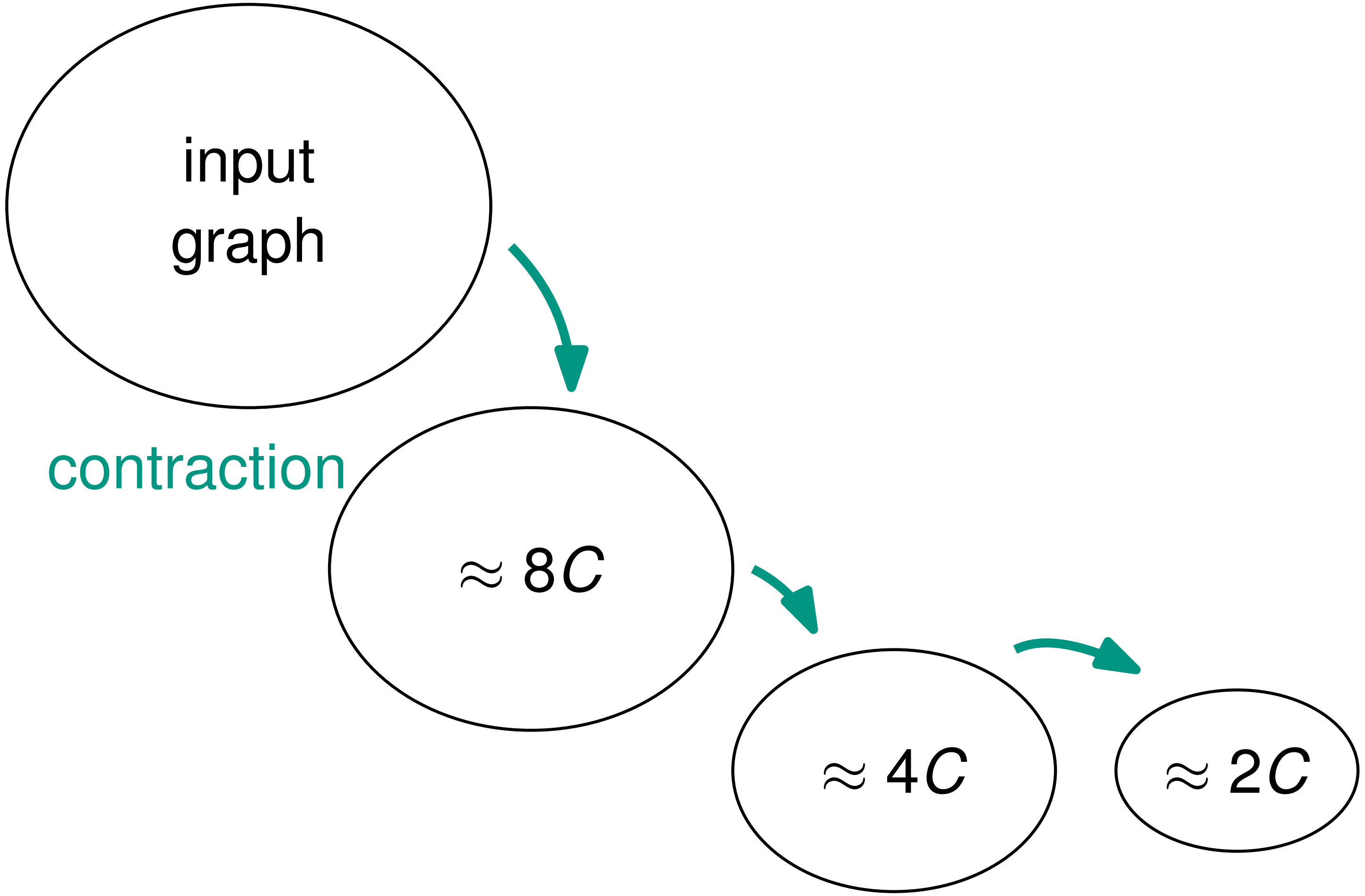
Deep Multilevel Graph Partitioning

- our contribution: **integrate coarsening into initial partitioning**



Deep Multilevel Graph Partitioning

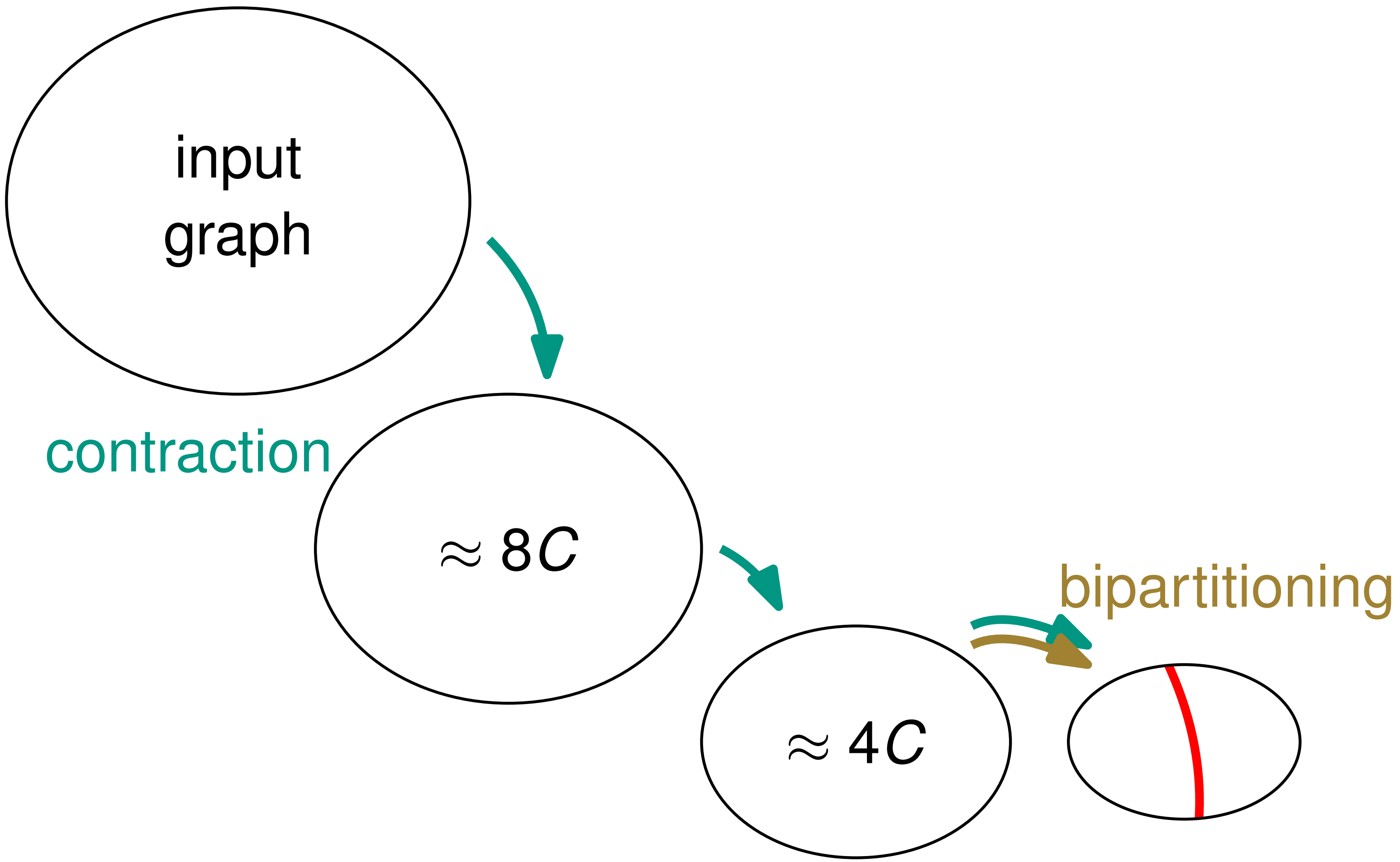
- our contribution: **integrate coarsening into initial partitioning**



Deep Multilevel Graph Partitioning

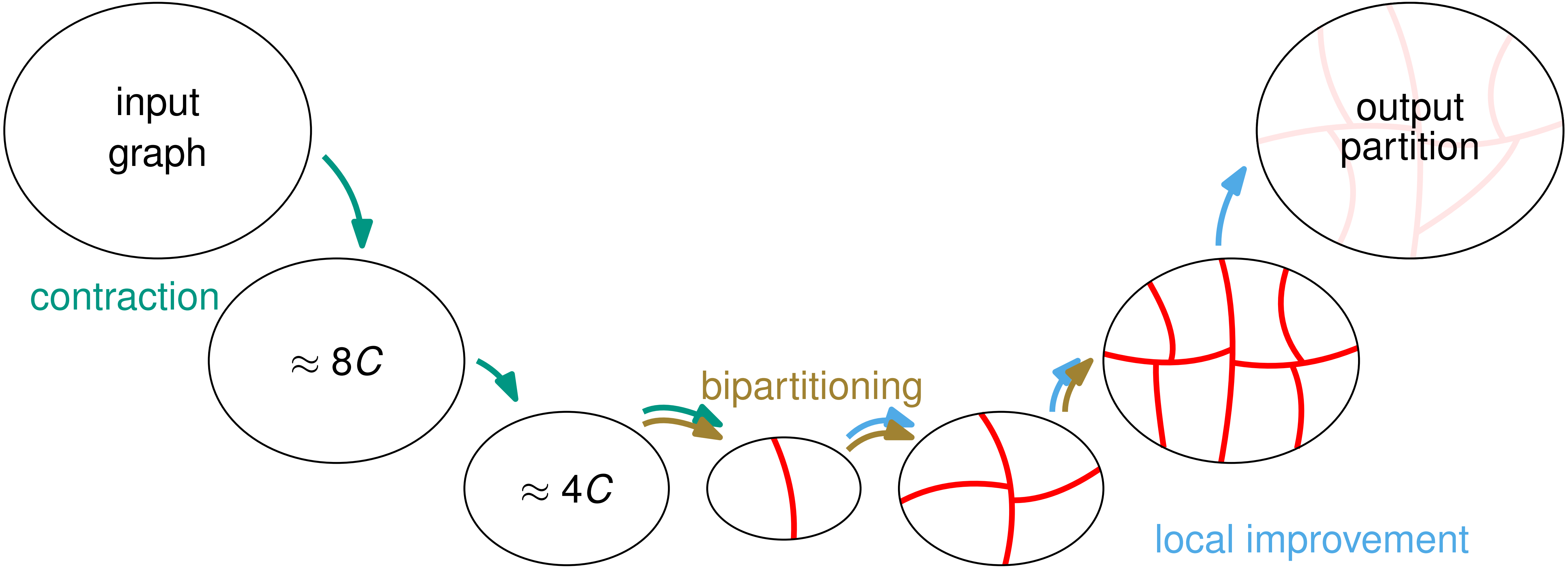
Karlsruhe Institute of Technology

- our contribution: **integrate coarsening into initial partitioning**



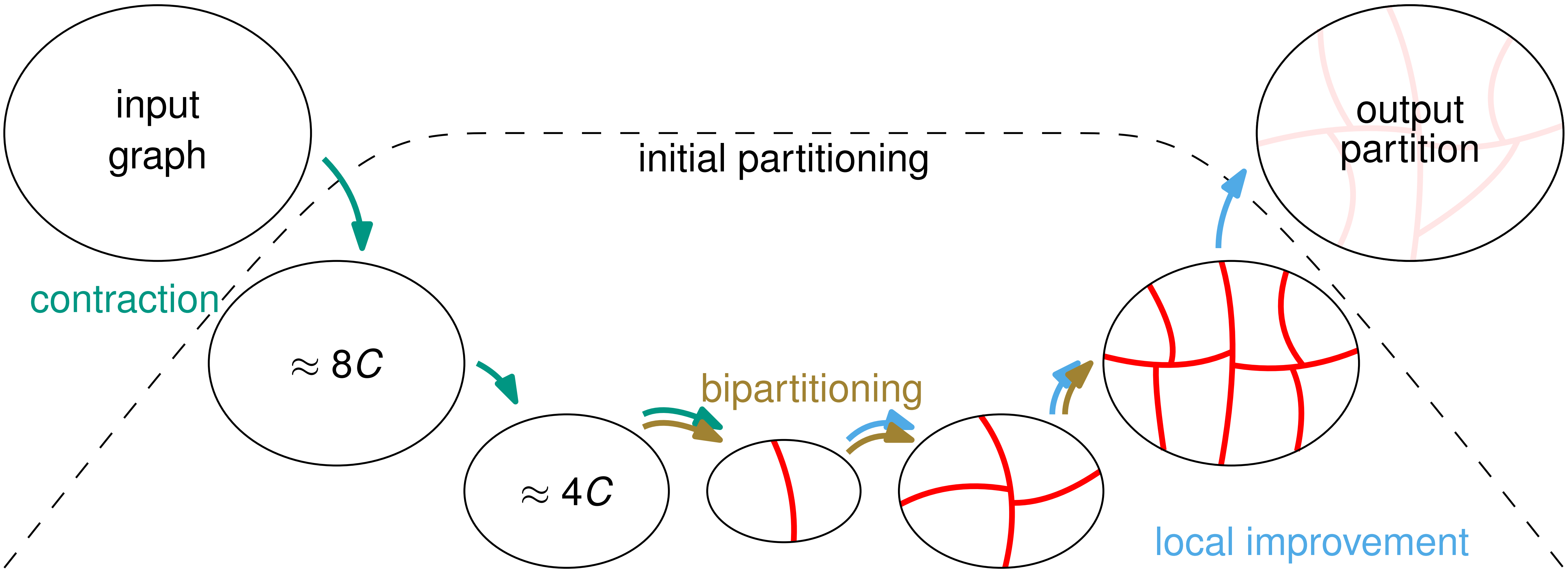
Deep Multilevel Graph Partitioning

- our contribution: **integrate coarsening into initial partitioning**



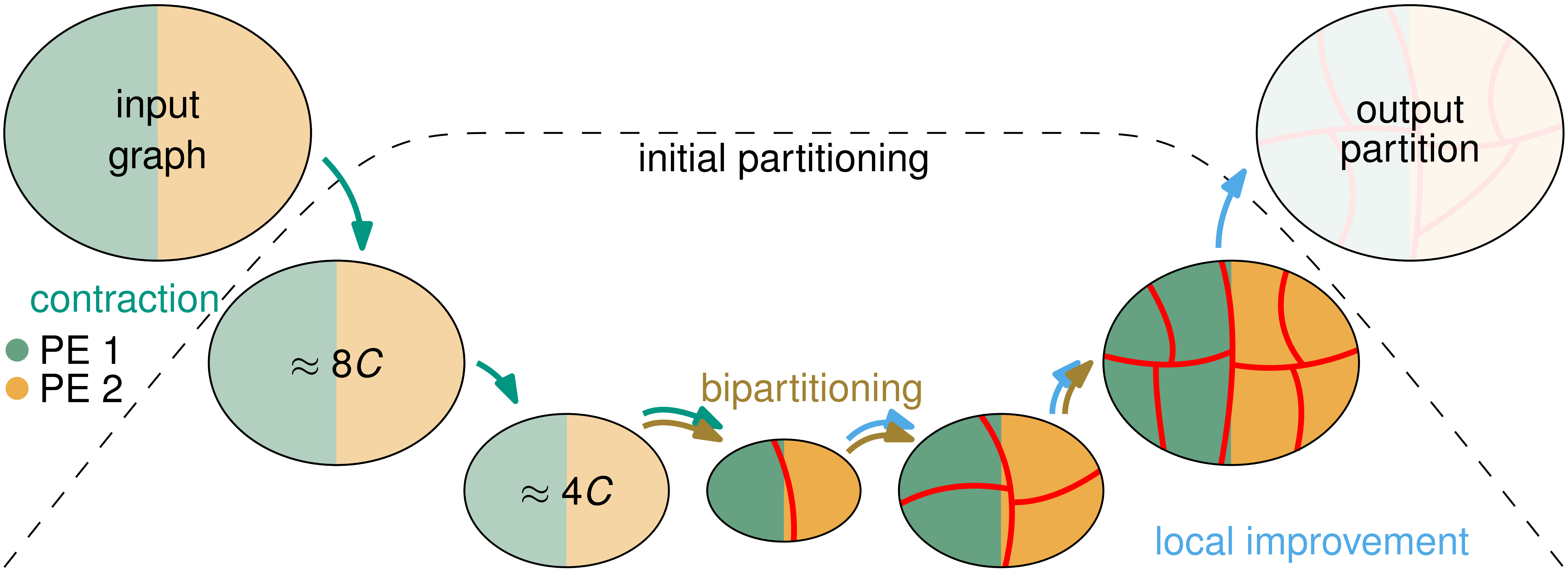
Deep Multilevel Graph Partitioning

- our contribution: **integrate coarsening into initial partitioning**



Deep Multilevel Graph Partitioning

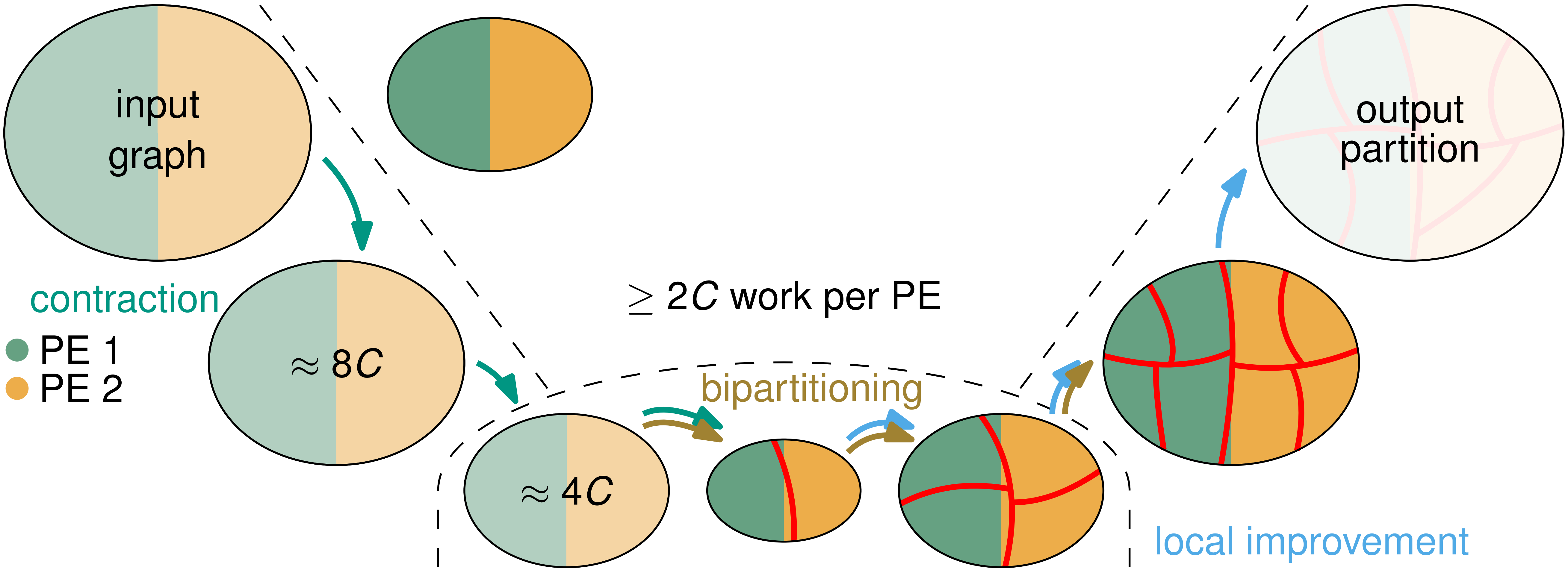
- our contribution: **integrate coarsening into initial partitioning**



Deep Multilevel Graph Partitioning

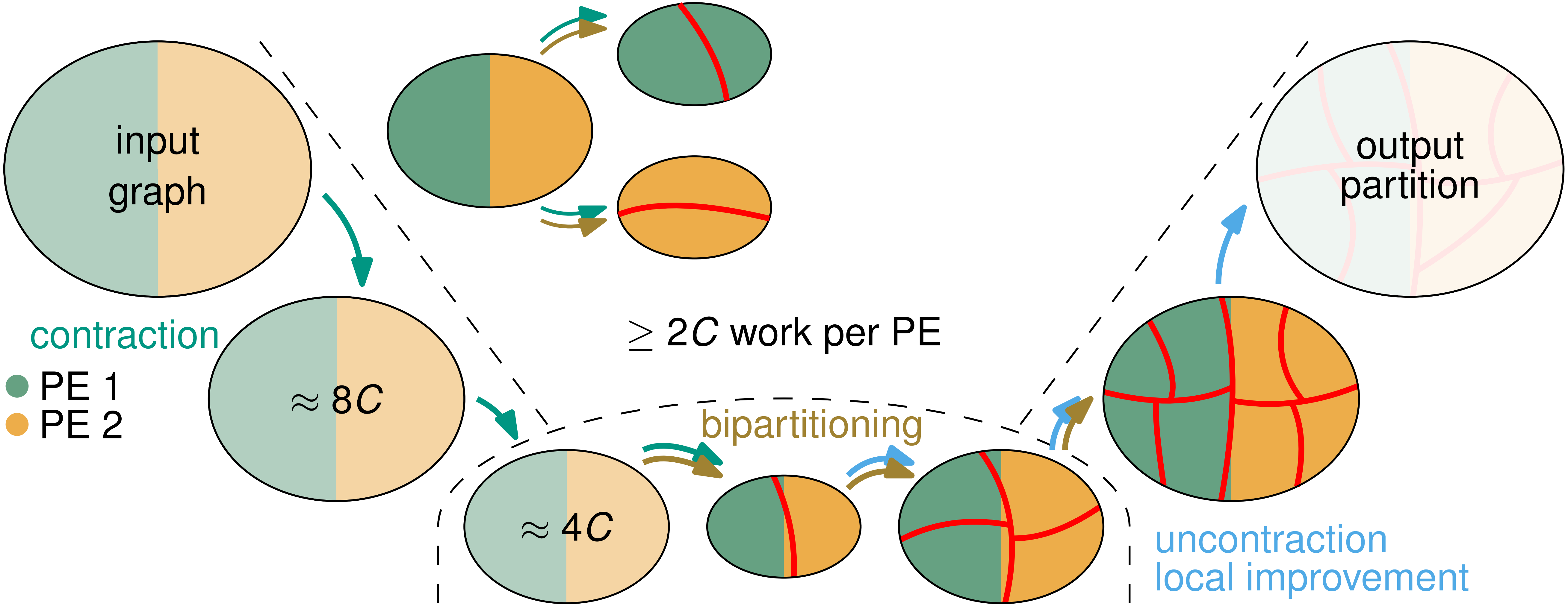
Karlsruhe Institute of Technology

- our contribution: **integrate coarsening into initial partitioning**



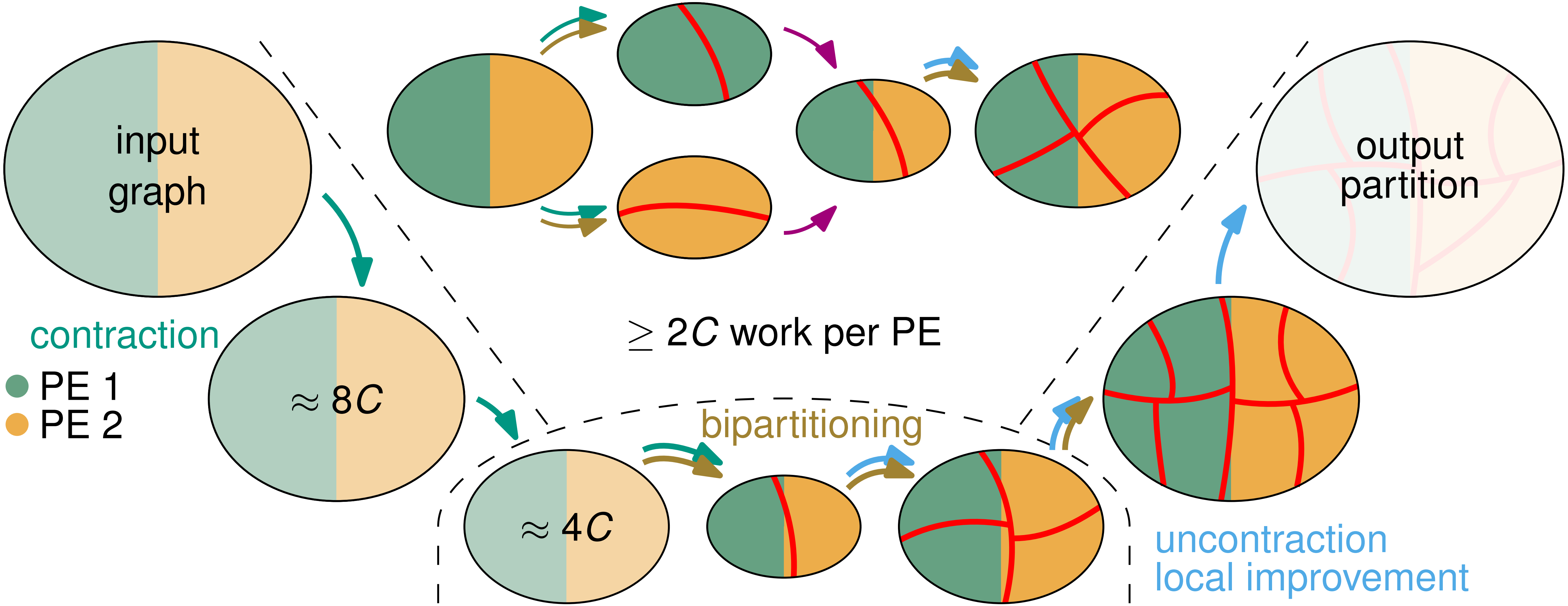
Deep Multilevel Graph Partitioning

- our contribution: **integrate coarsening into initial partitioning**



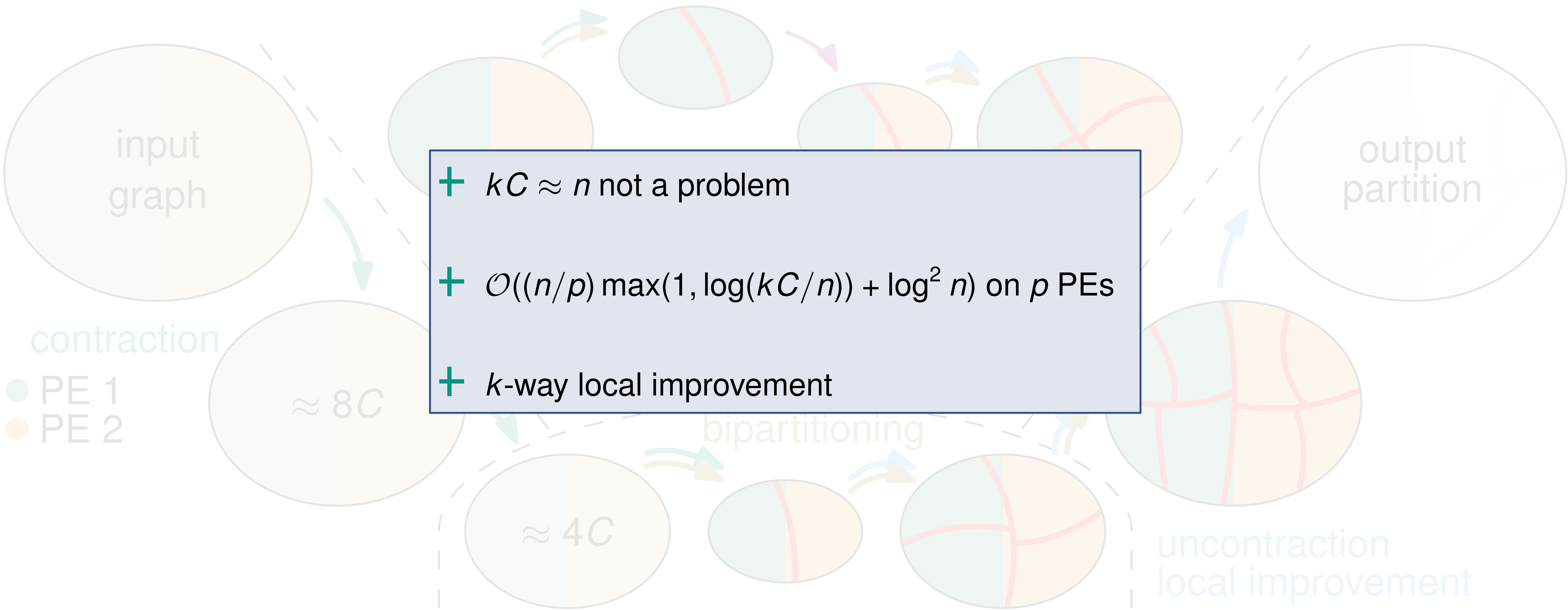
Deep Multilevel Graph Partitioning

- our contribution: **integrate coarsening into initial partitioning**



Deep Multilevel Graph Partitioning

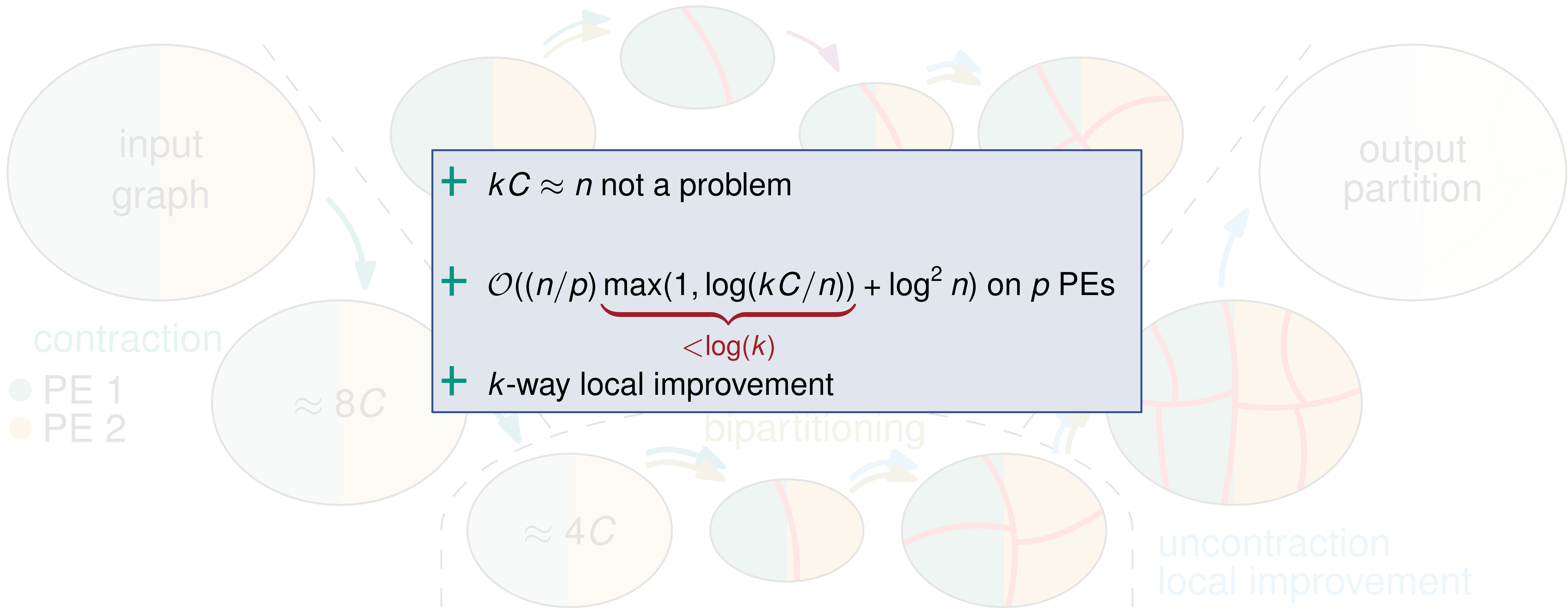
- our contribution: **integrate coarsening into initial partitioning**



+ $kC \approx n$ not a problem
+ $\mathcal{O}((n/p)\max(1, \log(kC/n)) + \log^2 n)$ on p PEs
+ k-way local improvement

Deep Multilevel Graph Partitioning

- our contribution: **integrate coarsening into initial partitioning**



- **+** $kC \approx n$ not a problem
- **+** $\mathcal{O}((n/p)\underbrace{\max(1, \log(kC/n))}_{<\log(k)} + \log^2 n)$ on p PEs
- **+** k-way local improvement

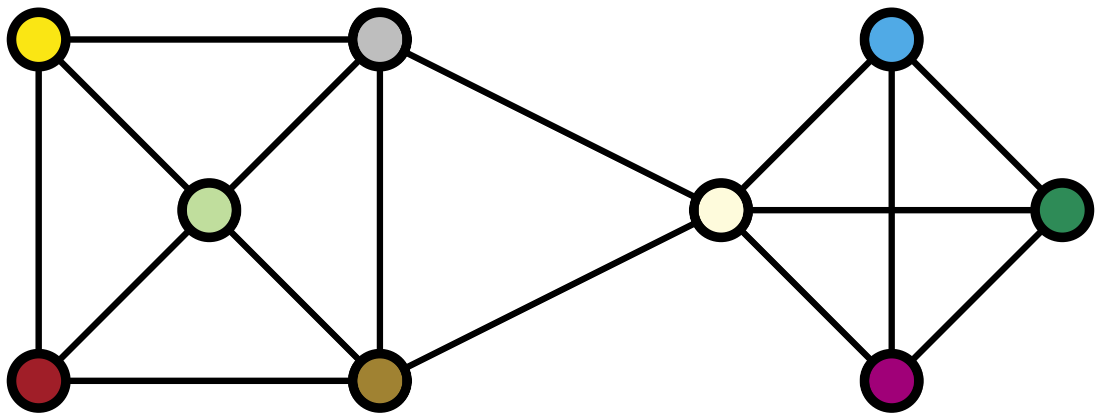
KaMinPar: Shared-memory Deep MGP

Karlsruhe Institute of Technology

- using **established** building-blocks for graph partitioning
  - **Coarsening:** size-constrained label propagation [Raghavan et al. 2007]
  - **Initial bipartitioning:** BFS + greedy graph growing + 2-way FM
  - **Uncoarsening:** size-constrained label propagation + balancer

KaMinPar: Shared-memory Deep MGP

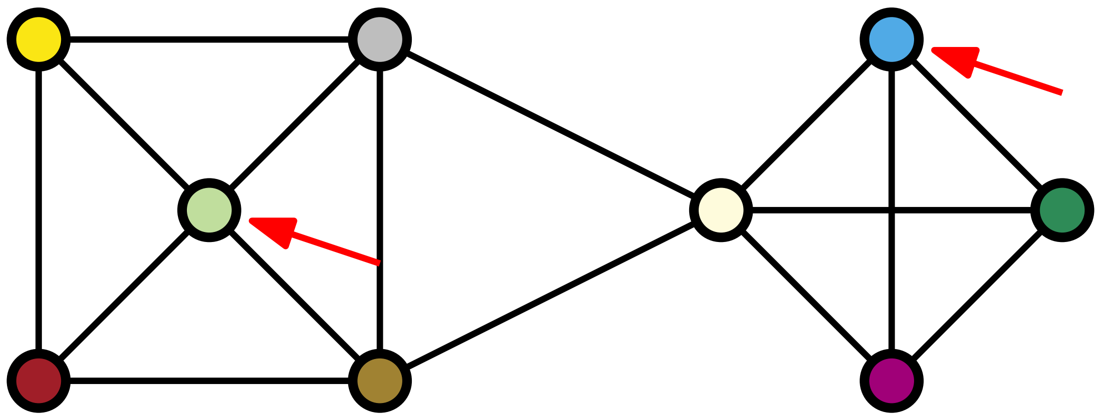
- using **established** building-blocks for graph partitioning
  - **Coarsening:** size-constrained label propagation [Raghavan et al. 2007]



  - **Initial bipartitioning:** BFS + greedy graph growing + 2-way FM
  - **Uncoarsening:** size-constrained label propagation + balancer

KaMinPar: Shared-memory Deep MGP

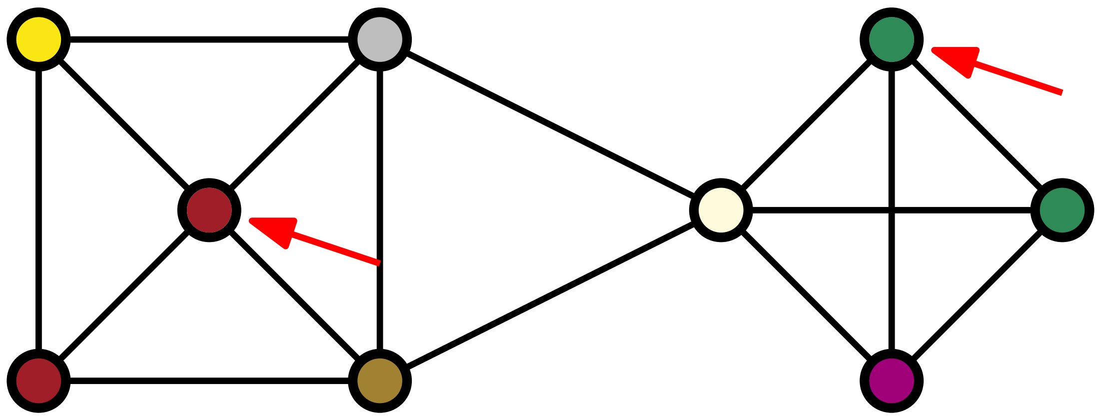
- using **established** building-blocks for graph partitioning
  - **Coarsening:** size-constrained label propagation [Raghavan et al. 2007]



  - **Initial bipartitioning:** BFS + greedy graph growing + 2-way FM
  - **Uncoarsening:** size-constrained label propagation + balancer

KaMinPar: Shared-memory Deep MGP

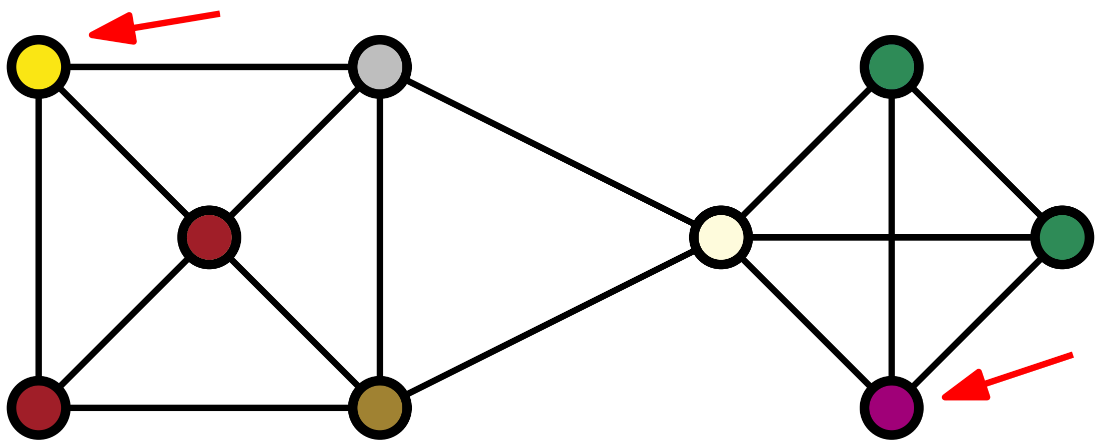
- using **established** building-blocks for graph partitioning
  - **Coarsening:** size-constrained label propagation [Raghavan et al. 2007]



  - **Initial bipartitioning:** BFS + greedy graph growing + 2-way FM
  - **Uncoarsening:** size-constrained label propagation + balancer

KaMinPar: Shared-memory Deep MGP

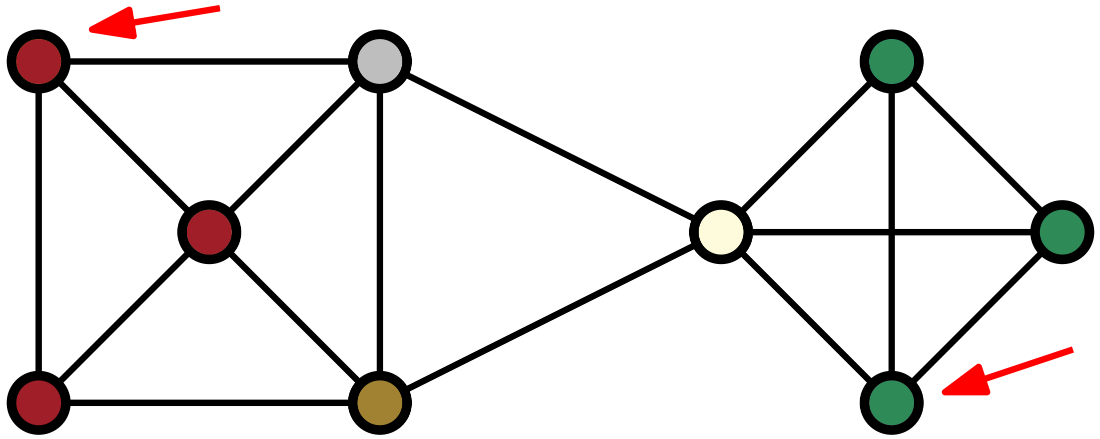
- using **established** building-blocks for graph partitioning
  - **Coarsening:** size-constrained label propagation [Raghavan et al. 2007]



  - **Initial bipartitioning:** BFS + greedy graph growing + 2-way FM
  - **Uncoarsening:** size-constrained label propagation + balancer

KaMinPar: Shared-memory Deep MGP

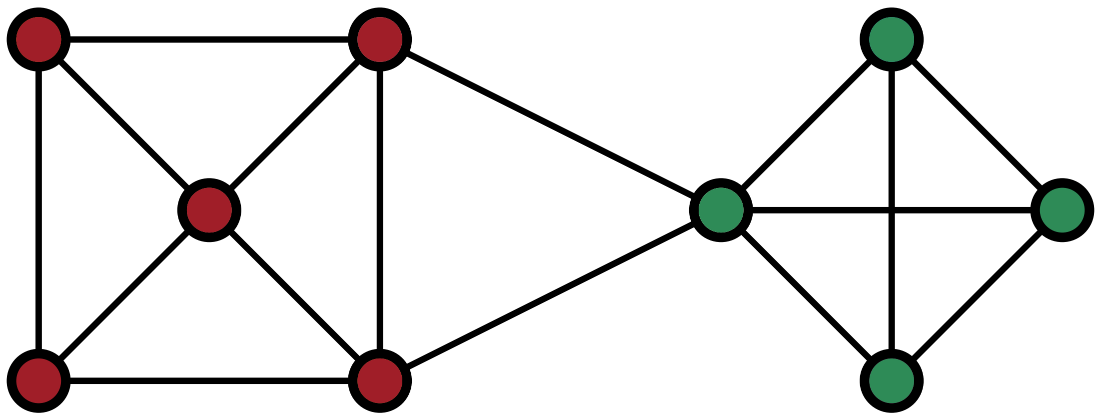
- using **established** building-blocks for graph partitioning
  - **Coarsening:** size-constrained label propagation [Raghavan et al. 2007]



  - **Initial bipartitioning:** BFS + greedy graph growing + 2-way FM
  - **Uncoarsening:** size-constrained label propagation + balancer

KaMinPar: Shared-memory Deep MGP

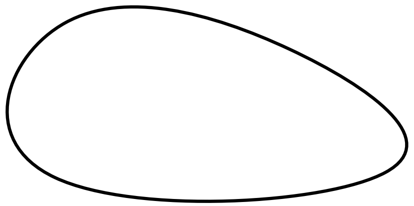
- using **established** building-blocks for graph partitioning
  - **Coarsening:** size-constrained label propagation [Raghavan et al. 2007]



  - **Initial bipartitioning:** BFS + greedy graph growing + 2-way FM
  - **Uncoarsening:** size-constrained label propagation + balancer

KaMinPar: Shared-memory Deep MGP

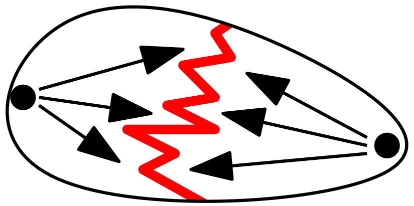
- using **established** building-blocks for graph partitioning
  - **Coarsening:** size-constrained label propagation [Raghavan et al. 2007]
  - **Initial bipartitioning:** BFS + greedy graph growing + 2-way FM



  - **Uncoarsening:** size-constrained label propagation + balancer

KaMinPar: Shared-memory Deep MGP

- using **established** building-blocks for graph partitioning
  - **Coarsening:** size-constrained label propagation [Raghavan et al. 2007]
  - **Initial bipartitioning:** BFS + greedy graph growing + 2-way FM

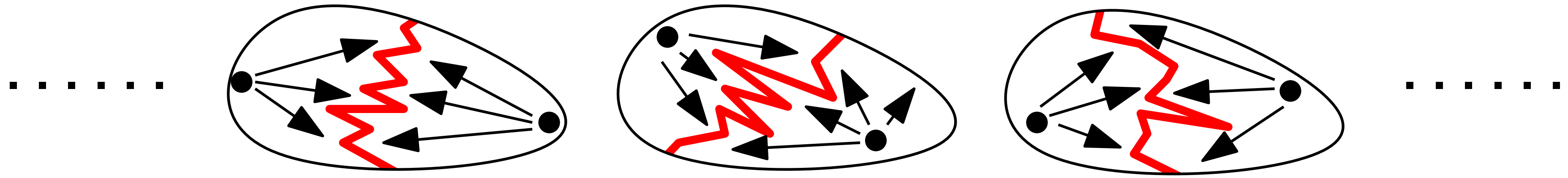


  - **Uncoarsening:** size-constrained label propagation + balancer

KaMinPar: Shared-memory Deep MGP

Karlsruhe Institute of Technology

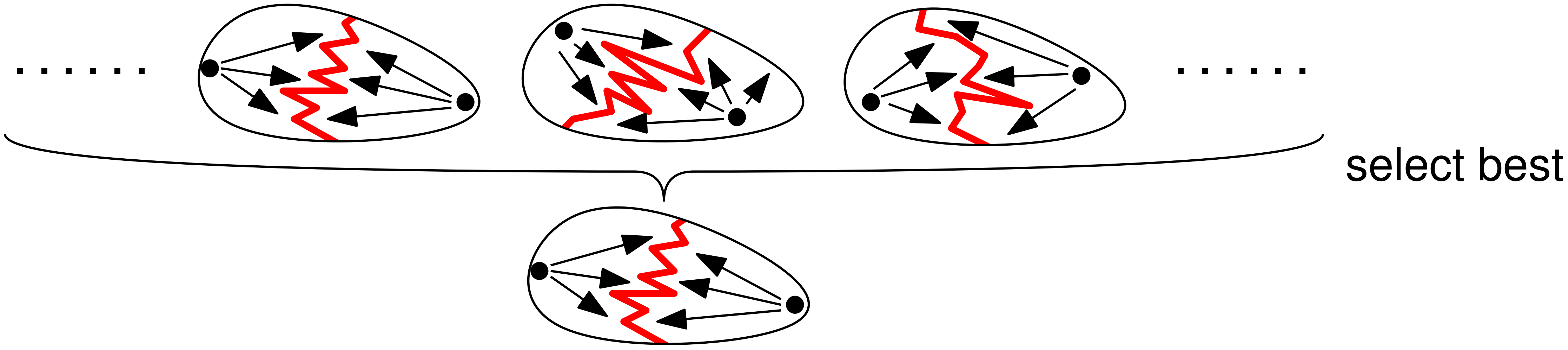
- using **established** building-blocks for graph partitioning
  - **Coarsening:** size-constrained label propagation [Raghavan et al. 2007]
  - **Initial bipartitioning:** BFS + greedy graph growing + 2-way FM



  - **Uncoarsening:** size-constrained label propagation + balancer

KaMinPar: Shared-memory Deep MGP

- using **established** building-blocks for graph partitioning
  - **Coarsening:** size-constrained label propagation [Raghavan et al. 2007]
  - **Initial bipartitioning:** BFS + greedy graph growing + 2-way FM



  - **Uncoarsening:** size-constrained label propagation + balancer

KaMinPar: Shared-memory Deep MGP

Karlsruhe Institute of Technology

- using **established** building-blocks for graph partitioning
  - **Coarsening:** size-constrained label propagation [Raghavan et al. 2007]
  - **Initial bipartitioning:** BFS + greedy graph growing + 2-way FM
  - **Uncoarsening:** size-constrained label propagation + balancer

Experiments – Benchmark Setup

- Scaling: up to 64 cores of 1 AMD EPYC 7702 @ 2 GHz, 1 TB RAM
- Comparison: 10 cores of 1 of 2 Intel Xeon Gold 6230 @ 2.1 GHz, 192 GB RAM

- Benchmark set: 21 large graphs
  - $100M \leq m \leq 1.8G$

- $k \in \{2^{11}, 2^{14}, 2^{17}, 2^{20}\}$

- Comparing **KaMinPar** with:
  - Mt-KaHiP
  - Mt-Metis-$\{$K, RB$\}$
  - PuLP

Shared-memory parallel

Experiments – Benchmark Setup

Karlsruhe Institute of Technology

- Scaling: up to 64 cores of 1 AMD EPYC 7702 @ 2 GHz, 1 TB RAM
- Comparison: 10 cores of 1 of 2 Intel Xeon Gold 6230 @ 2.1 GHz, 192 GB RAM

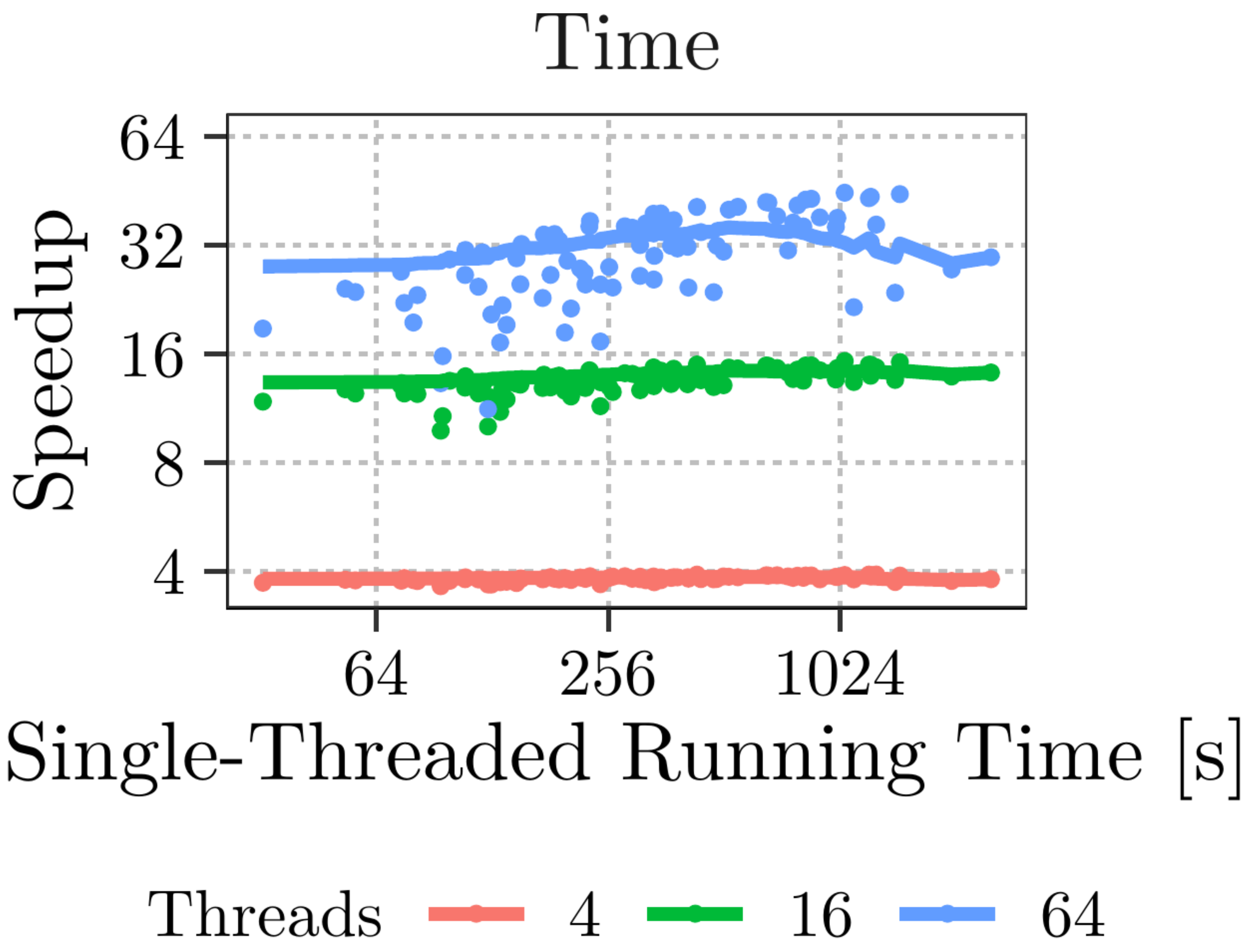
- Benchmark set: 21 large graphs
  - $100M \leq m \leq 1.8G$

- $k \in \{2^{11}, 2^{14}, 2^{17}, 2^{20}\}$ ”large” values of k

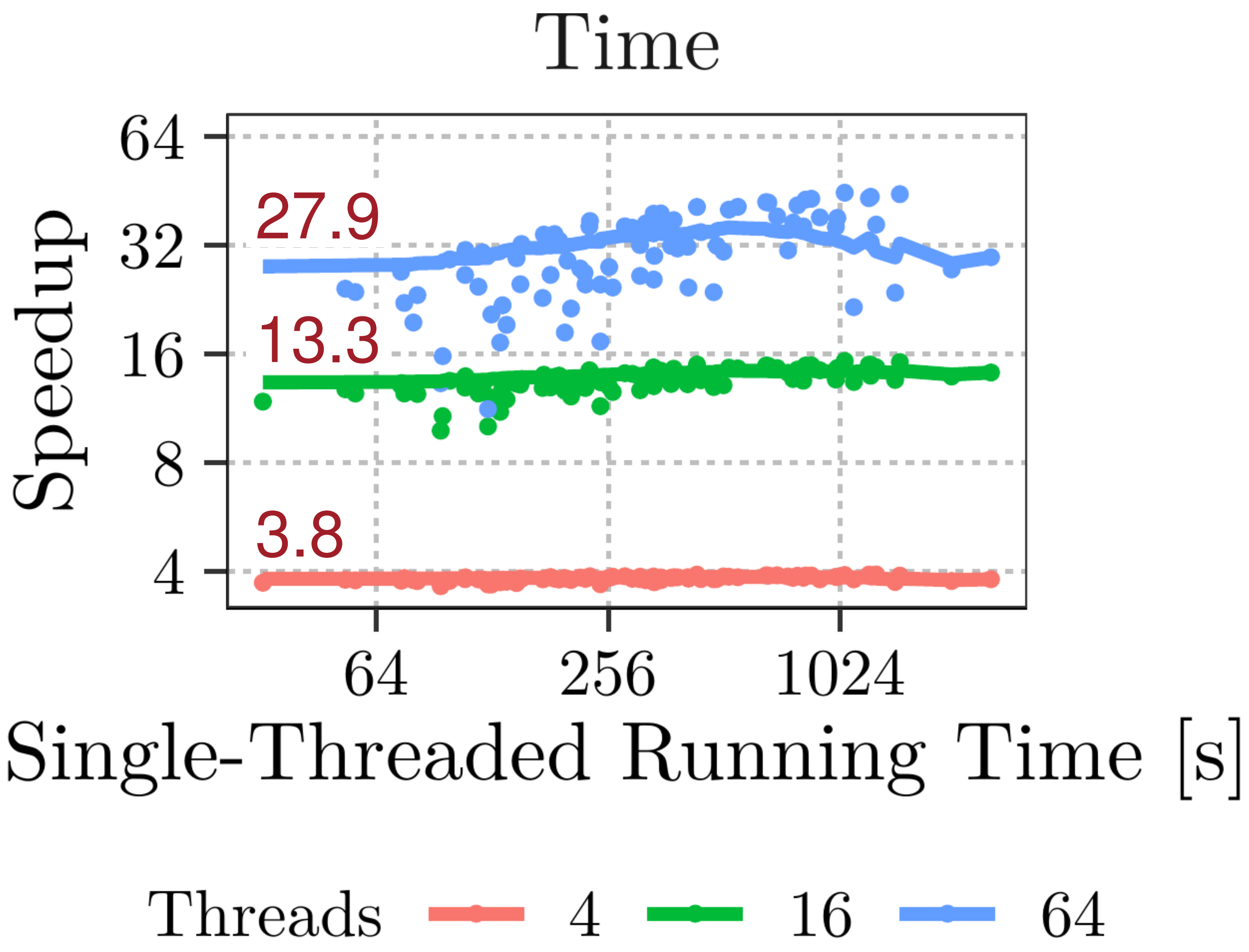
- Comparing **KaMinPar** with:
  - Mt-KaHiP
  - Mt-Metis-{K, RB}
  - PuLP

  Shared-memory parallel

Experiments – Results



Experiments – Results



Experiments – Results

Algorithm	# timeout	# crash	# imbalanced	# feasible	rel. time	rel. cut
KaMinPar	0%	0%	0%	**100%**	**1.00**	1.00
Mt-Metis-K	23%	12%	61%	5%	11.91	**0.99**
Mt-Metis-RB	0%	30%	65%	5%	5.61	1.03
Mt-KaHiP	37%	8%	13%	42%	38.64	1.00
PuLP	90%	0%	0%	10%	73.52	1.25

84 instances on 10 cores

Experiments – Results

Algorithm	# timeout	# crash	# imbalanced	# feasible	rel. time	rel. cut
KaMinPar	0%	0%	0%	**100%**	**1.00**	1.00
Mt-Metis-K	23%	12%	61%	5%	11.91	**0.99**
Mt-Metis-RB	0%	30%	65%	5%	5.61	1.03
Mt-KaHiP	37%	8%	13%	42%	38.64	1.00
PuLP	90%	0%	0%	10%	73.52	1.25

84 instances on 10 cores

Experiments – Results

Karlsruhe Institute of Technology

Time limit = 1 *h*

Algorithm	# timeout	# crash	# imbalanced	# feasible	rel. time	rel. cut
KaMinPar	0%	0%	0%	**100%**	**1.00**	1.00
Mt-Metis-K	23%	12%	61%	5%	11.91	**0.99**
Mt-Metis-RB	0%	30%	65%	5%	5.61	1.03
Mt-KaHiP	37%	8%	13%	42%	38.64	1.00
PuLP	90%	0%	0%	10%	73.52	1.25

84 instances on 10 cores

Experiments – Results

Time limit = 1 h

Running time $<$ 6 min

Algorithm	# timeout	# crash	# imbalanced	# feasible	rel. time	rel. cut
KaMinPar	0%	[illegible]	[illegible]%	**100%**	**1.00**	1.00
Mt-Metis-K	23%	12%	61%	5%	11.91	**0.99**
Mt-Metis-RB	0%	30%	65%	5%	5.61	1.03
Mt-KaHiP	37%	8%	13%	42%	38.64	1.00
PuLP	90%	0%	0%	10%	73.52	1.25

84 instances on 10 cores

Experiments – Results

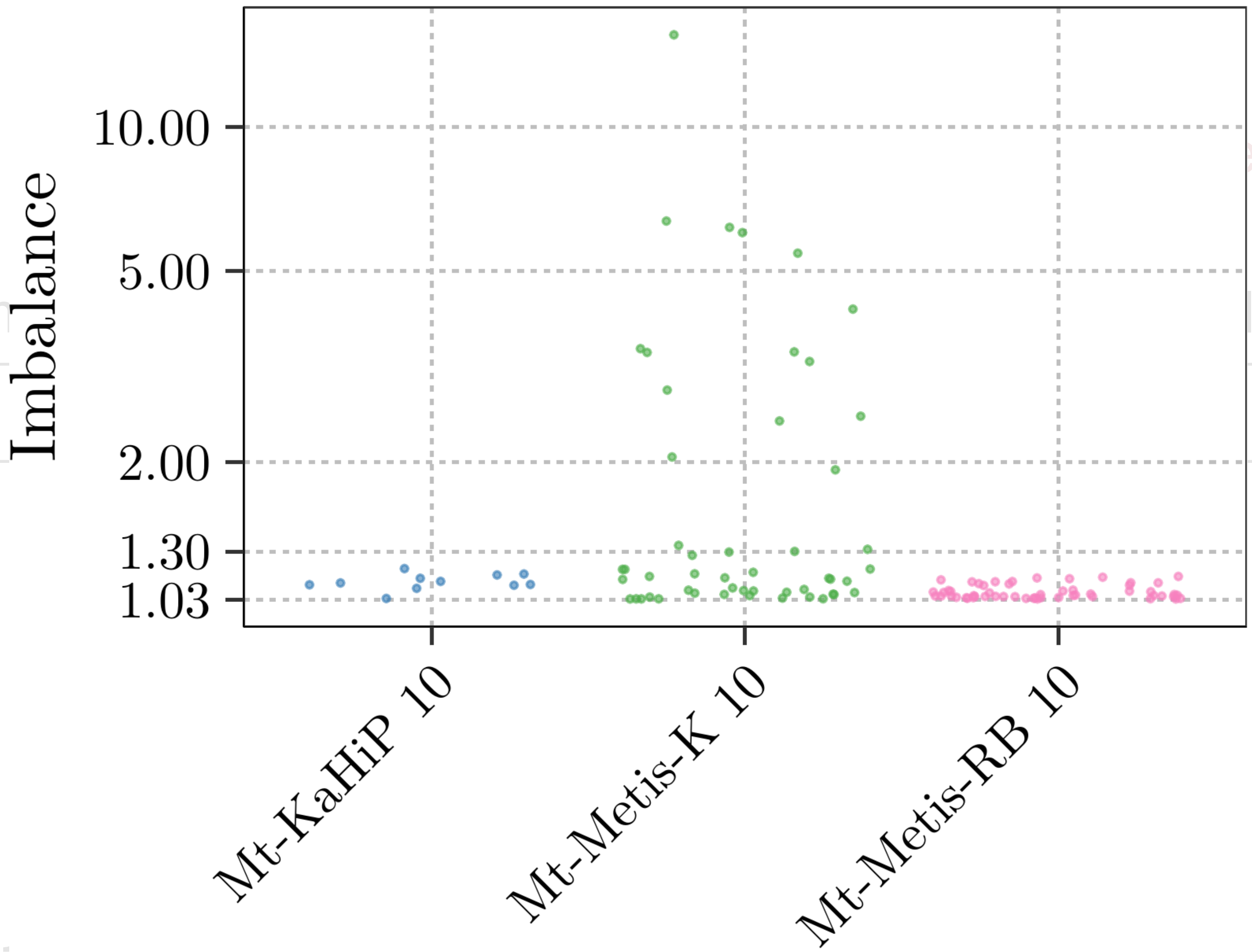
Karlsruhe Institute of Technology

Imbalance $>$ 3%

Algorithm	# timeout	# crash	# imbalanced	# feasible	rel. time	rel. cut
KaMinPar	0%	0%	0%	**100%**	**1.00**	1.00
Mt-Metis-K	23%	12%	61%	5%	11.91	**0.99**
Mt-Metis-RB	0%	30%	65%	5%	5.61	1.03
Mt-KaHiP	37%	8%	13%	42%	38.64	1.00
PuLP	90%	0%	0%	10%	73.52	1.25

84 instances on 10 cores

Experiments – Results



e > 3%

Algorithm	# ti	l. time	rel. cut
KaMinPar		**1.00**	1.00
Mt-Metis-K		11.91	**0.99**
Mt-Metis-RB		5.61	1.03
Mt-KaHiP		38.64	1.00
PuLP		73.52	1.25

84 instances on 10 cores

Experiments – Results

Karlsruhe Institute of Technology

Algorithm	# timeout	# crash	# imbalanced	# feasible	rel. time	rel. cut
KaMinPar	0%	0%	0%	**100%**	**1.00**	1.00
Mt-Metis-K	23%	12%	61%	5%	11.91	**0.99**
Mt-Metis-RB	0%	30%	65%	5%	5.61	1.03
Mt-KaHiP	37%	8%	13%	42%	38.64	1.00
PuLP	90%	0%	0%	10%	73.52	1.25

orders of magnitude faster
vs direkt *k*-way

84 instances on 10 cores

Experiments – Results

Karlsruhe Institute of Technology

Algorithm	# timeout	# crash	# imbalanced	# feasible	rel. time	rel. cut
KaMinPar	0%	0%	0%	**100%**	**1.00**	1.00
Mt-Metis-K	23%	12%	61%	5%	11.91	**0.99**
Mt-Metis-RB	0%	30%	65%	5%	5.61	1.03
Mt-KaHiP	37%	8%	13%	42%	38.64	1.00
PuLP	90%	0%	0%	10%	73.52	1.25

84 instances on 10 cores

Experiments – Results

Algorithm	# timeout	# crash	# imbalanced	# feasible	rel. time	rel. cut
KaMinPar	0%	0%	0%	**100%**	**1.00**	1.00
Mt-Metis-K	23%	12%	61%	5%	11.91	**0.99**
Mt-Metis-RB	0%	30%	65%	5%	5.61	1.03
Mt-KaHiP	37%	8%	13%	42%	38.64	1.00
PuLP	90%	0%	0%	10%	73.52	1.25

84 instances on 10 cores

Experiments – Benchmark Setup

Karlsruhe Institute of Technology

- System: 10 cores of 1 of 2 Intel Xeon Gold 6230 @ 2.1 GHz, 96 GB RAM
- Benchmark set: 197 graphs (1 k $\leq m \leq$ 1.8 G)
- $k \in \{2, 4, 8, 16, 32, 64\}$
- Comparing **KaMinPar** with:
  - Mt-KaHiP
  - Mt-Metis
  - PuLP

  Shared-memory parallel

  - KaHiP-fsocial
  - Metis

  Sequential (paper only)

Experiments – Benchmark Setup

Karlsruhe Institute of Technology

- System: 10 cores of 1 of 2 Intel Xeon Gold 6230 @ 2.1 GHz, 96 GB RAM
- Benchmark set: 197 graphs (1 k $\leq m \leq$ 1.8 G)
- $k \in \{2, 4, 8, 16, 32, 64\}$ ”normal” values of k
- Comparing **KaMinPar** with:
  - Mt-KaHiP
  - Mt-Metis
  - PuLP

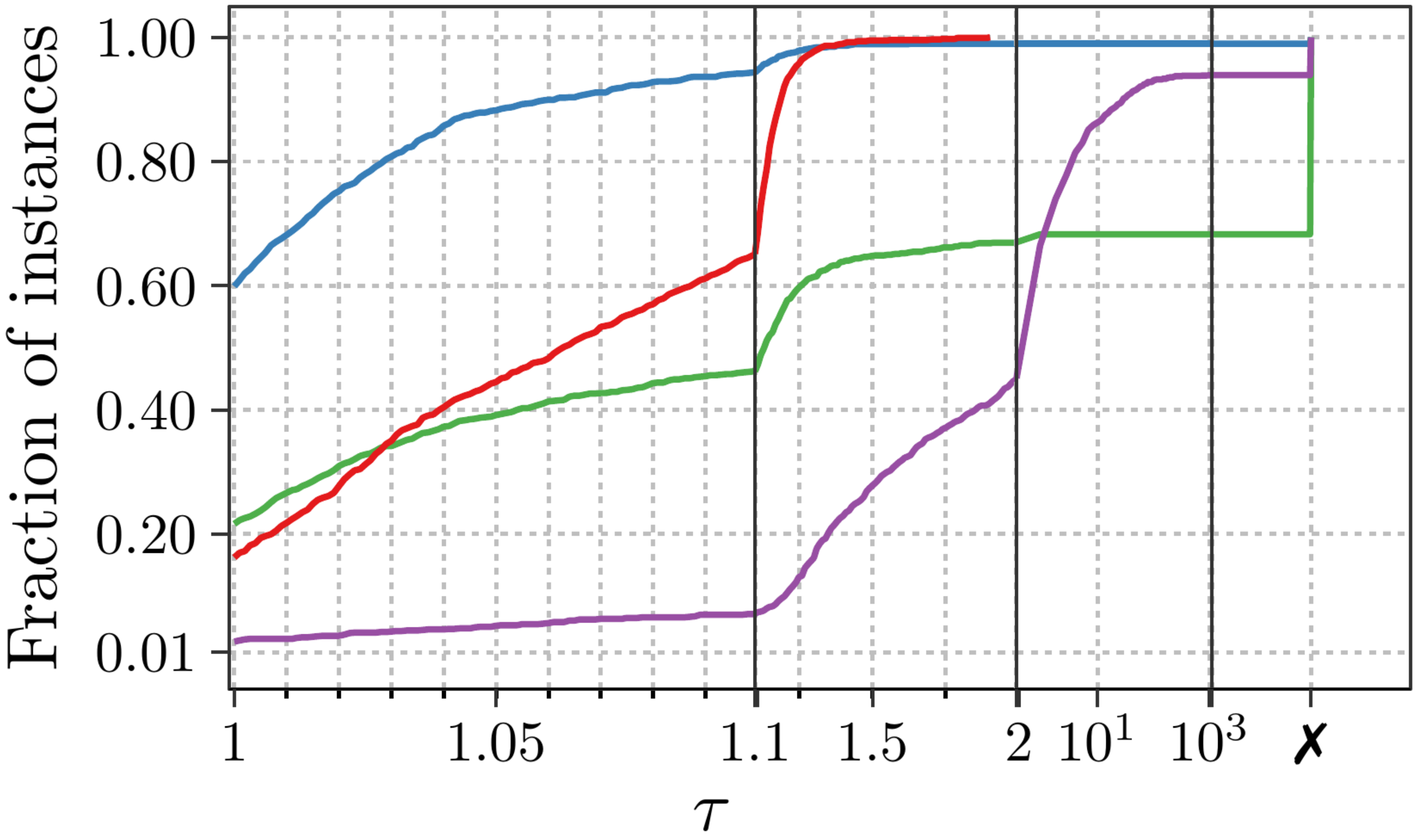
  Shared-memory parallel

  - KaHiP-fsocial
  - Metis

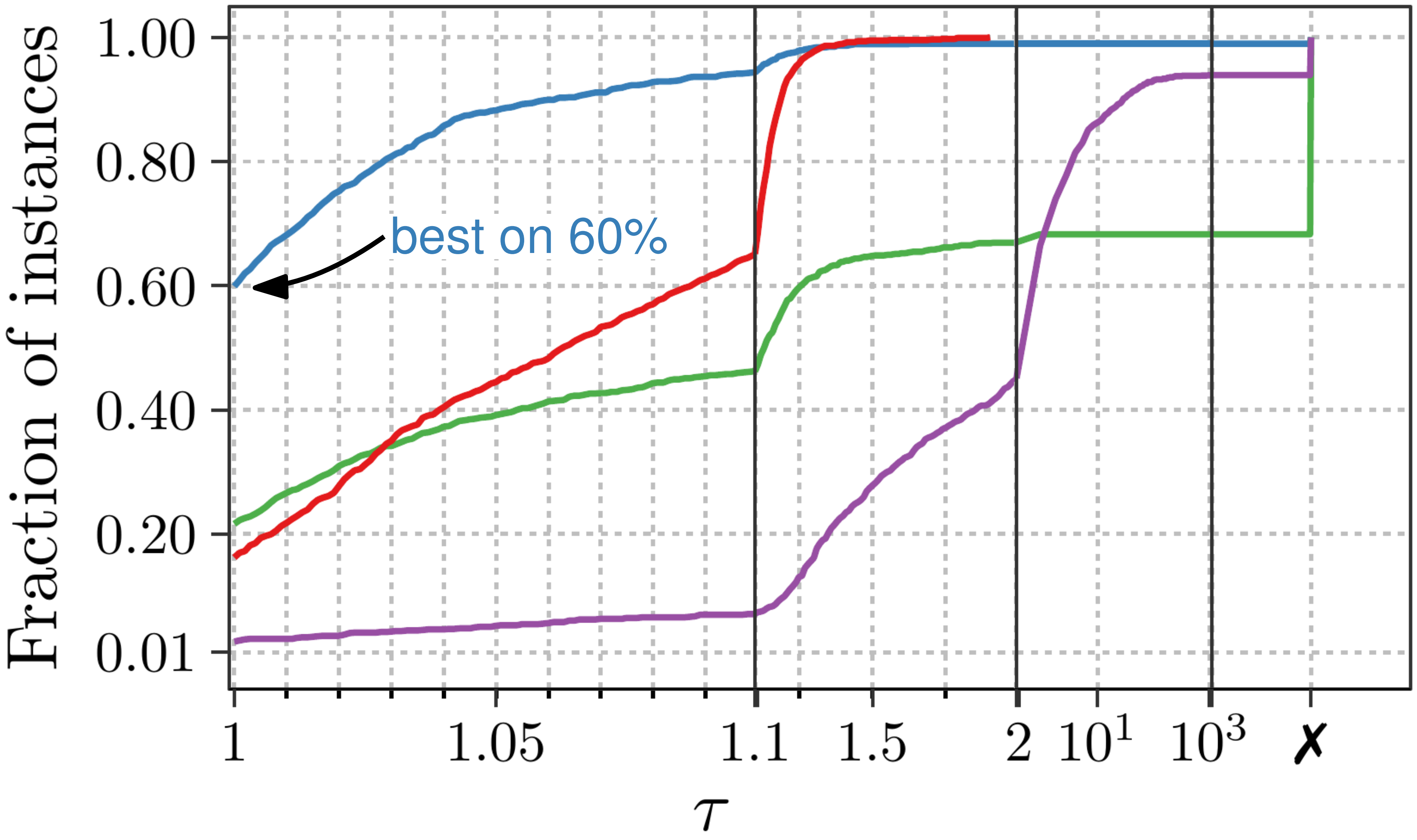
  Sequential (paper only)

Experiments – Edge Cut

Karlsruhe Institute of Technology

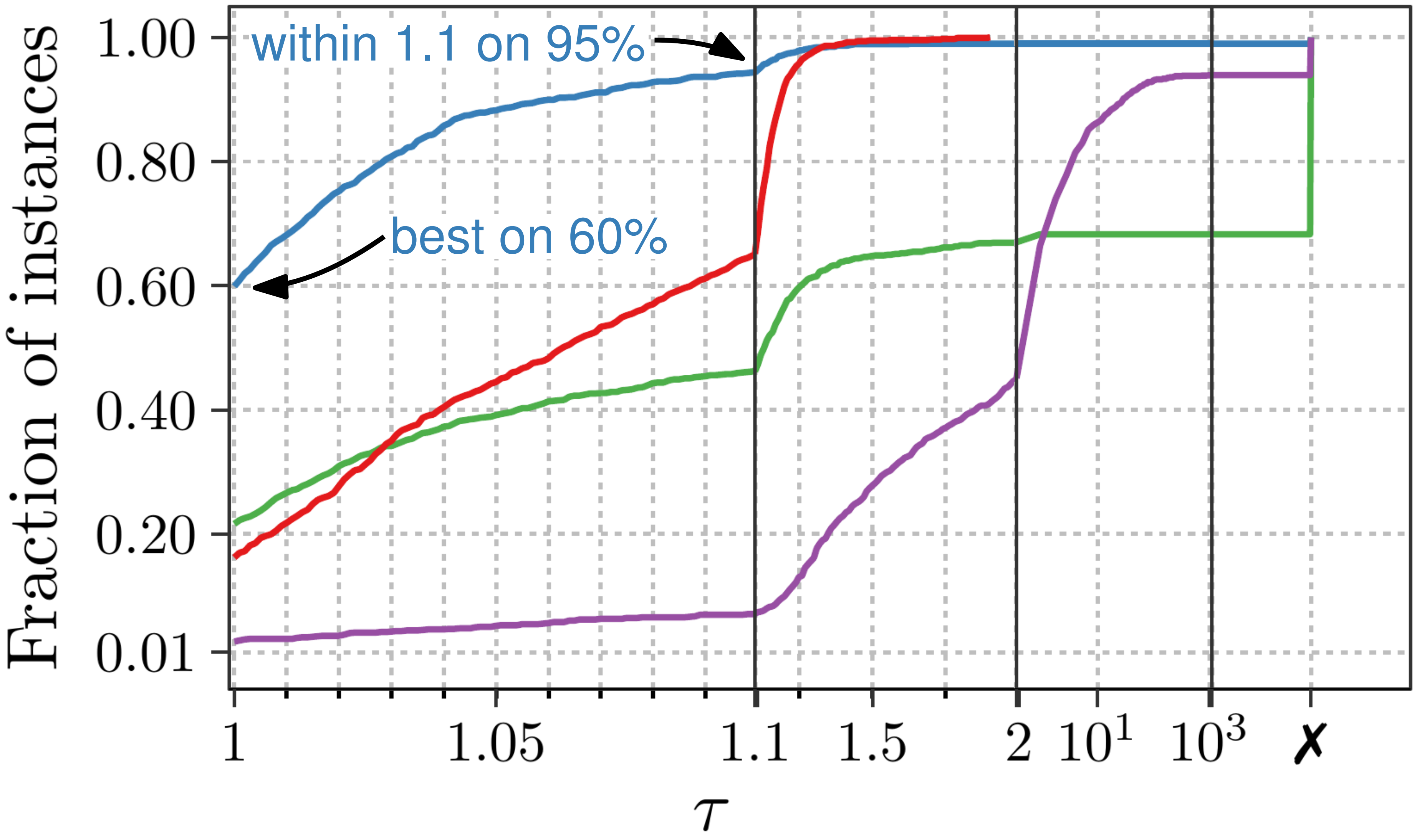


Experiments – Edge Cut

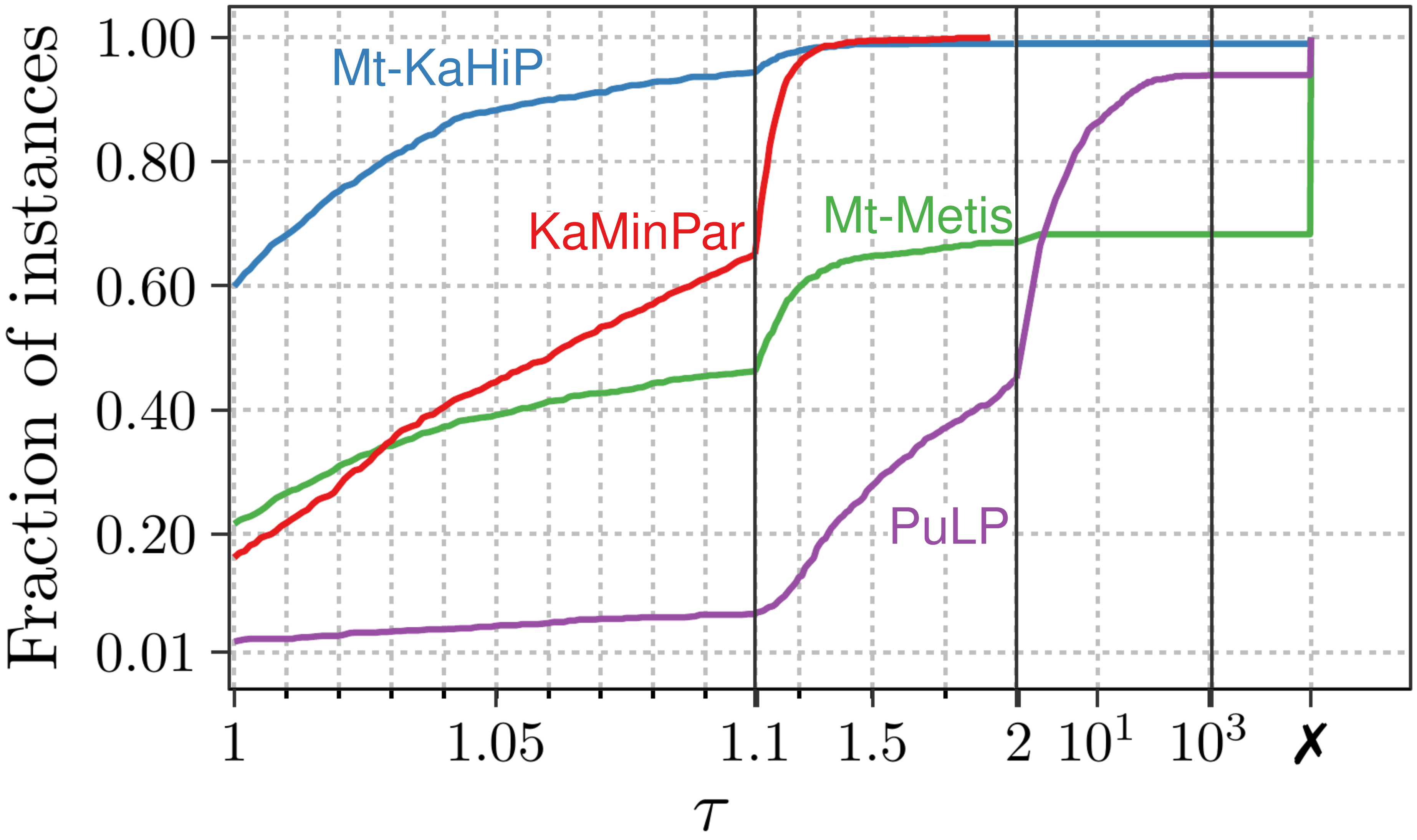


Experiments – Edge Cut

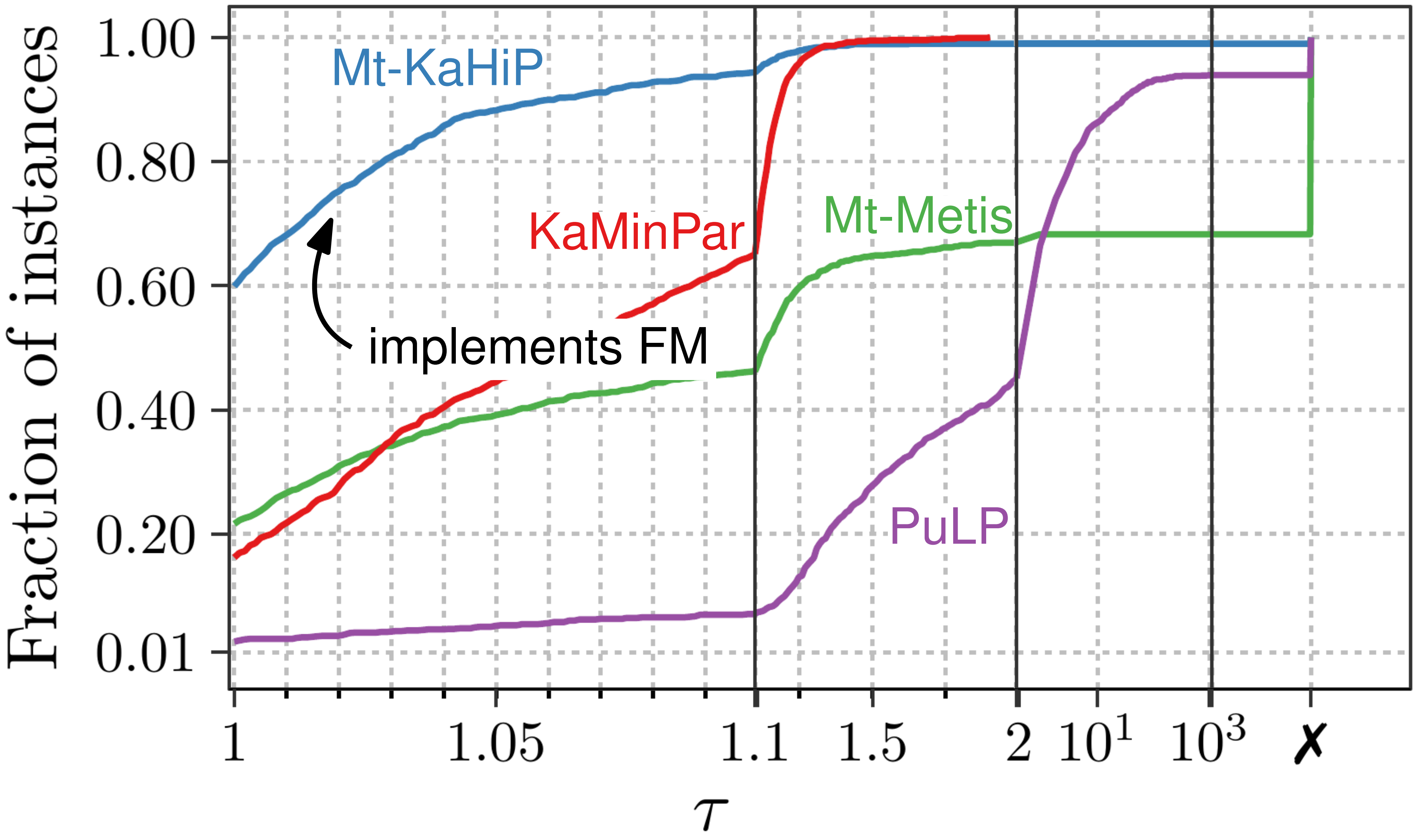
Karlsruhe Institute of Technology



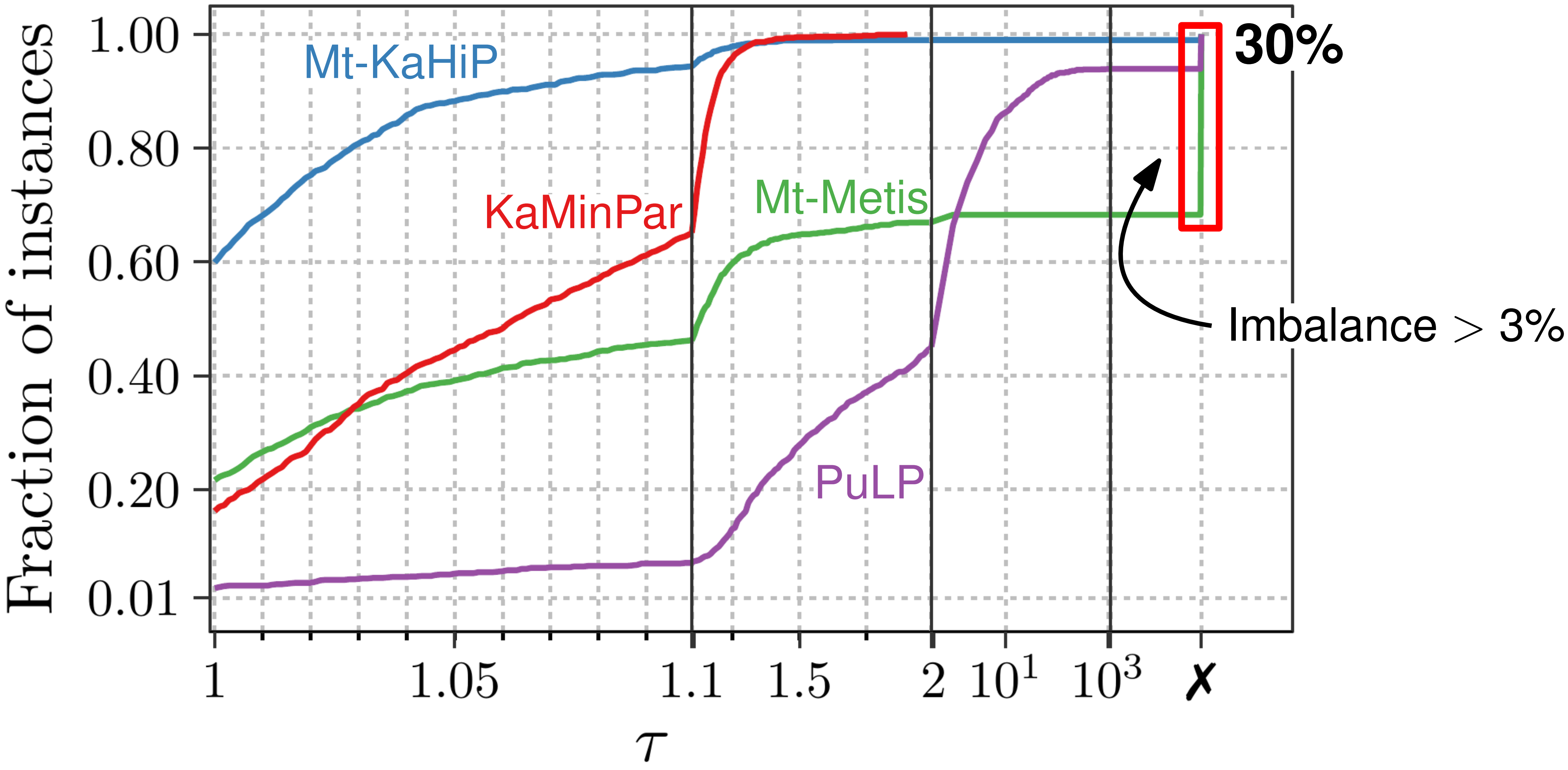
Experiments – Edge Cut



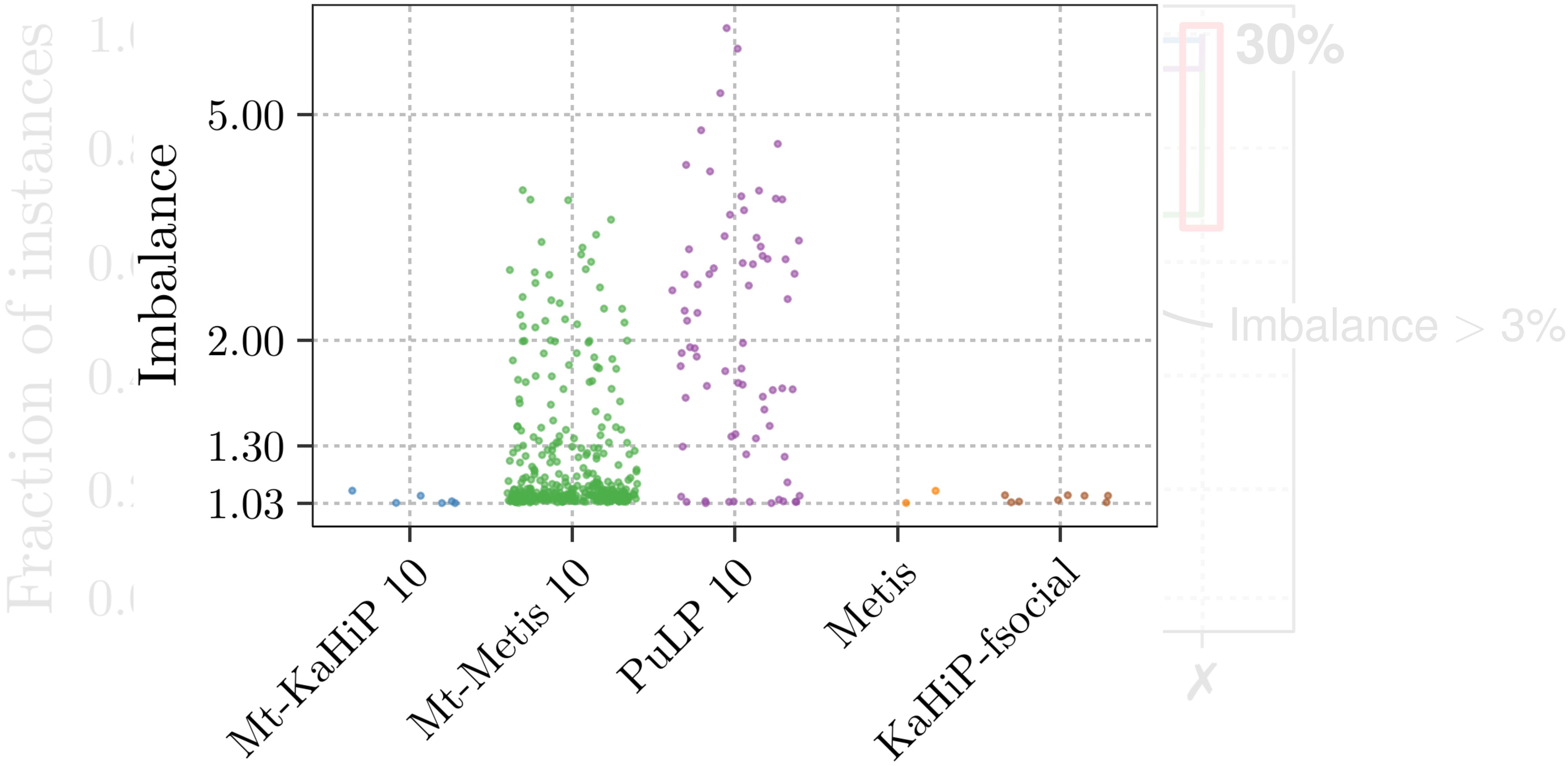
Experiments – Edge Cut



Experiments – Edge Cut



Experiments – Edge Cut



Experiments – Running Time

Karlsruhe Institute of Technology

Algorithm	T	$T[m \geq 10^6]$	$T[m \geq 10^8]$	rel. cut	# i
KaMinPar 10	**0.39 s**	**0.85 s**	**9.36 s**	1.00	
Mt-Metis 10	0.48 s	1.49 s	30.36 s	1.00	
Mt-KaHiP 10	1.33 s	3.84 s	55.76 s	**0.94**	
PuLP 10	1.11 s	5.70 s	95.93 s	2.39	
Metis	1.00 s	4.15 s	97.44 s	1.05	
KaHiP-fsocial	2.93 s	11.05 s	200.67 s	1.03	
# instances	1,150	832	196		

Conclusion

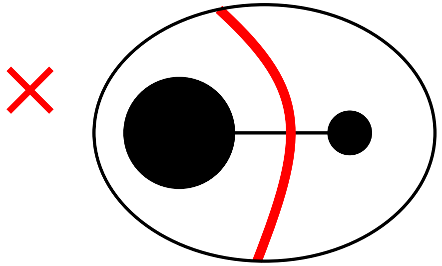
- Deep Multilevel Graph Partitioning:
  - Integrate coarsening deep into initial partitioning
- **KaMinPar**: deep MGP implementation
  - Order of magnitude faster for large k than competing tools
  - Comparable to competing tools for small k
- **Future**: limits of MGP, $k = O(n)$ – parallel FM – distributed DMGP
- Supplementary data available online:
  - Full experimental results: `algo2.iti.kit.edu/seemaier/deep_mgp/`
  - Source code: `github.com/KaHIP/KaMinPar`

Maintaining the Balance Constraint

- ”Standard” balance constraint: $c(V_i) \leq (1 + \varepsilon)\lceil \frac{c(V)}{k} \rceil$
  - Problem: NP-complete for general node weights

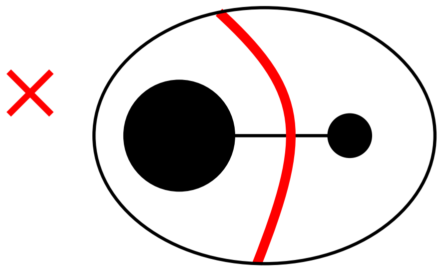
Maintaining the Balance Constraint

- "Standard" balance constraint: $c(V_i) \leq (1 + \varepsilon)\lceil \frac{c(V)}{k} \rceil$
  - Problem: NP-complete for general node weights



Maintaining the Balance Constraint

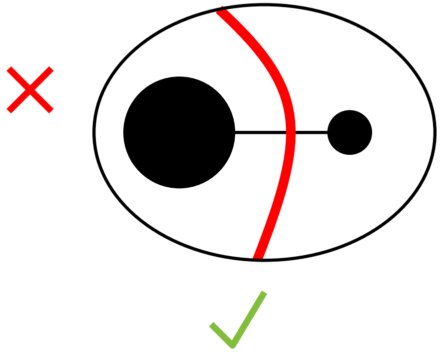
- ”Standard” balance constraint: $c(V_i) \leq (1+\varepsilon)\lceil \frac{c(V)}{k} \rceil$
  - Problem: NP-complete for general node weights



- Our approach: relax to $c(V_i) \leq \max((1+\varepsilon)\frac{c(V)}{k}, \frac{c(V)}{k} + \max_v c(v))$
  - the good: trivial to satisfy
  - the bad: uncontraction changes $\max_v c(v)$
    - but: violation bounded by $\max_v c(v)$

Maintaining the Balance Constraint

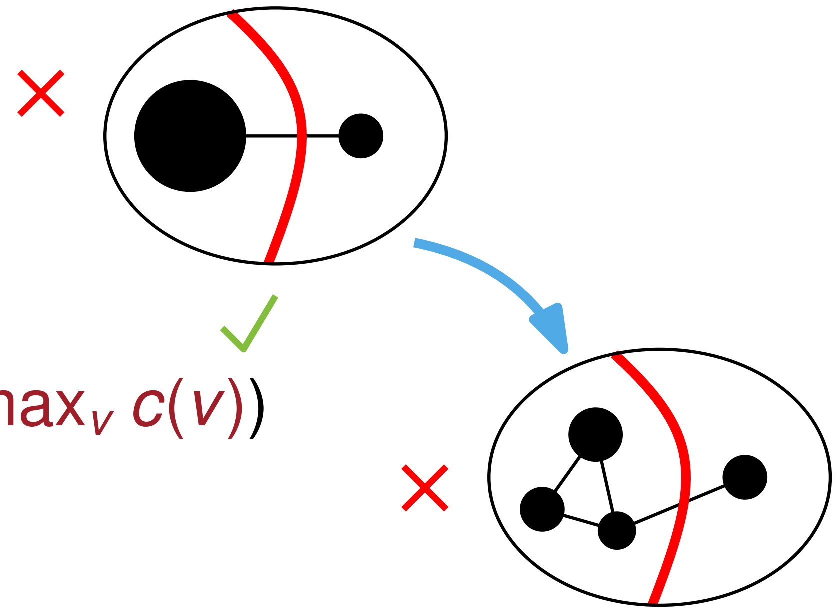
- "Standard" balance constraint: $c(V_i) \leq (1+\varepsilon)\lceil \frac{c(V)}{k} \rceil$
  - Problem: NP-complete for general node weights



- Our approach: relax to $c(V_i) \leq \max((1+\varepsilon)\frac{c(V)}{k}, \frac{c(V)}{k} + \max_v c(v))$
  - the good: trivial to satisfy
  - the bad: uncontraction changes $\max_v c(v)$
    - but: violation bounded by $\max_v c(v)$

Maintaining the Balance Constraint

- ”Standard” balance constraint: $c(V_i) \leq (1+\varepsilon)\lceil \frac{c(V)}{k} \rceil$
  - Problem: NP-complete for general node weights
- Our approach: relax to $c(V_i) \leq \max((1+\varepsilon)\frac{c(V)}{k}, \frac{c(V)}{k} + \max_v c(v))$
  - the good: trivial to satisfy
  - the bad: uncontraction changes $\max_v c(v)$
    - but: violation bounded by $\max_v c(v)$



Maintaining the Balance Constraint

- "Standard" balance constraint: $c(V_i) \leq (1+\varepsilon)\lceil \frac{c(V)}{k} \rceil$
  - Problem: NP-complete for general node weights
- Our approach: relax to $c(V_i) \leq \max((1+\varepsilon)\frac{c(V)}{k}, \frac{c(V)}{k} + \max_v c(v))$
  - the good: trivial to satisfy
  - the bad: uncontraction changes $\max_v c(v)$
    - but: violation bounded by $\max_v c(v)$