hpcgarage.org/ppam15

Algorithmic time, energy, and power trade-offs

in graph computations (?)

Sara Karamati · **Jeff Young** · **Jee Choi [IBM Research]** · **Richard (Rich) Vuduc**
· **Oded Green** *[Bader lab @ GT]* · **Marat Dukhan** · **Anita Zakrzewska** *[Bader lab @ GT]*

September 7, 2015 — 11th Int'l. Conf. Parallel Processing and Applied Mathematics (PPAM) — http://ppam.pl

Georgia Tech College of Computing
Computational Science and Engineering

hpcgarage

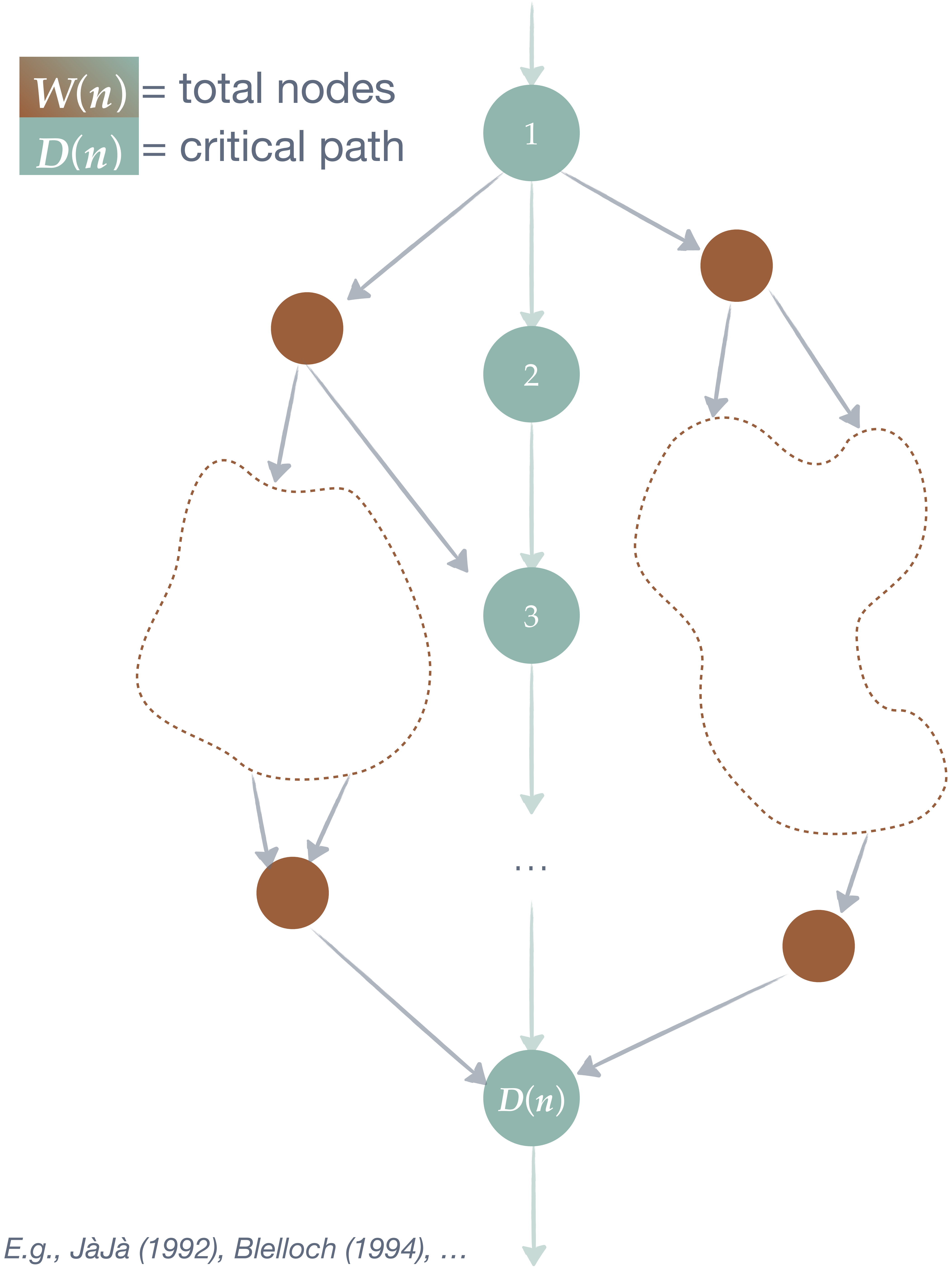

Kraków
bound!

Do we need **new principles of algorithm design** when optimizing **energy** or **power** instead of **time** (or storage)?

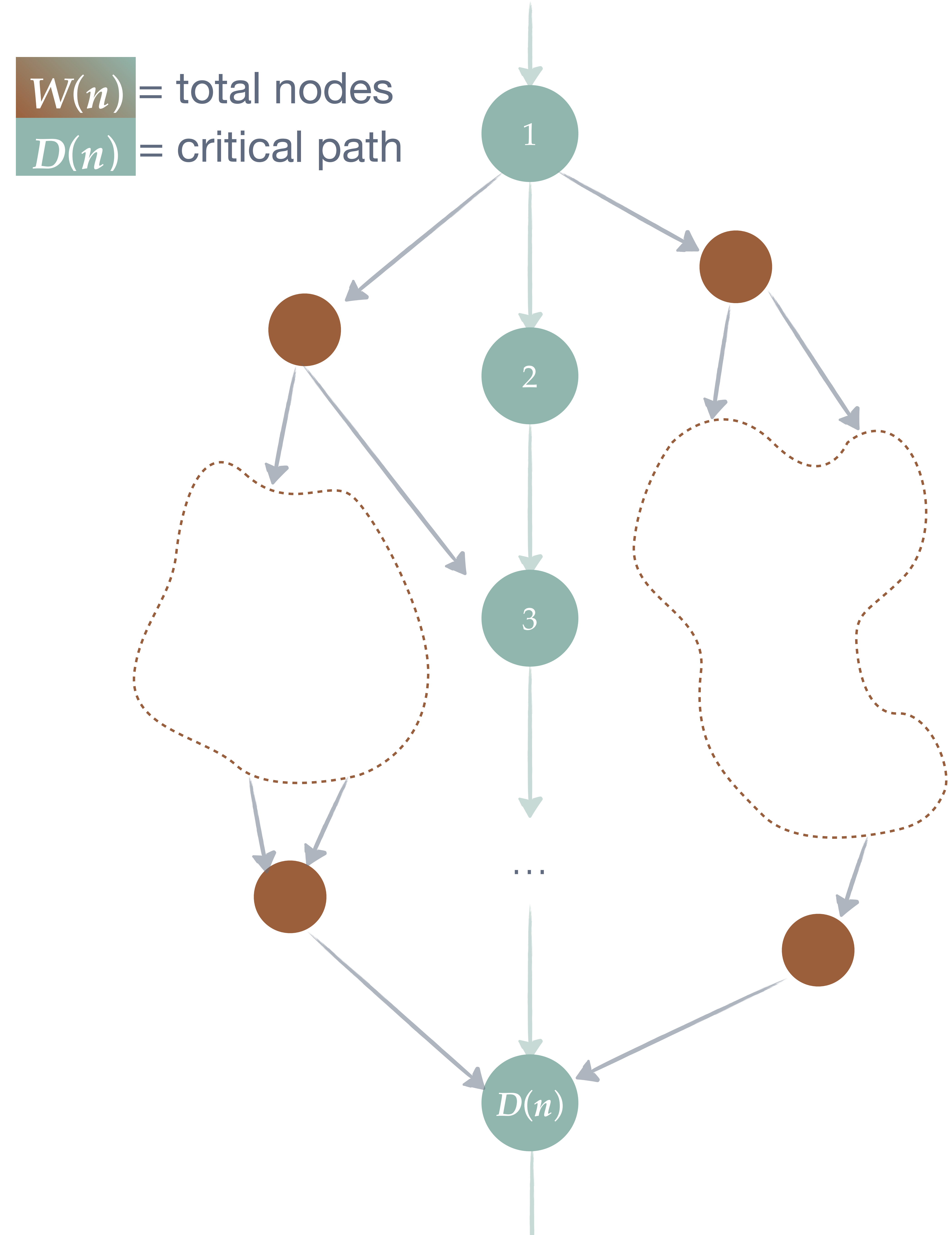
Do we need **new principles of algorithm design** when optimizing **energy** or **power** instead of **time** (or storage)?

My answer: **No** — We have the main principles, but they are *even more important* when the metric changes, especially to energy instead of time.

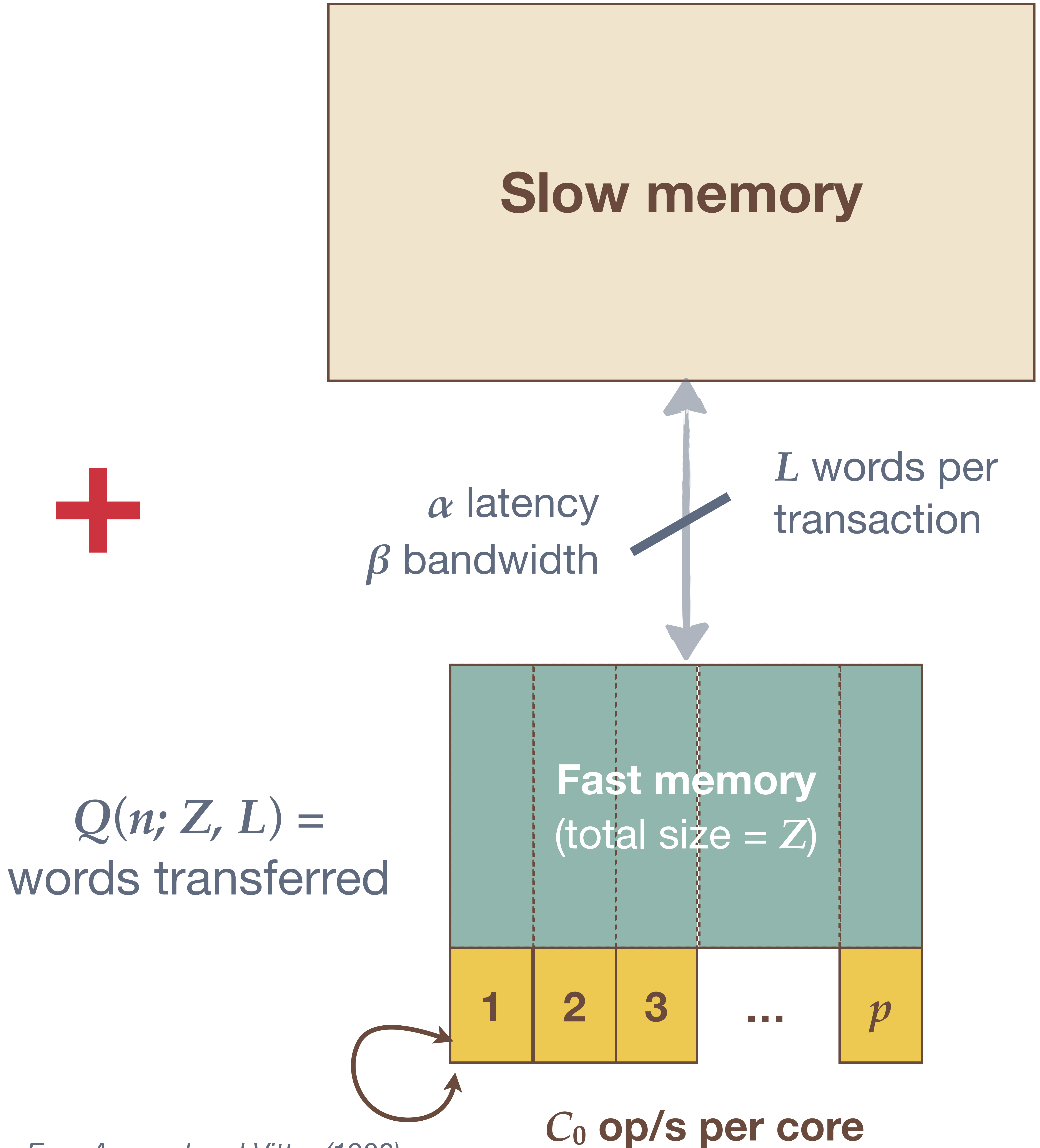
I. Basic abstract models



E.g., JàJà (1992), Blelloch (1994), …

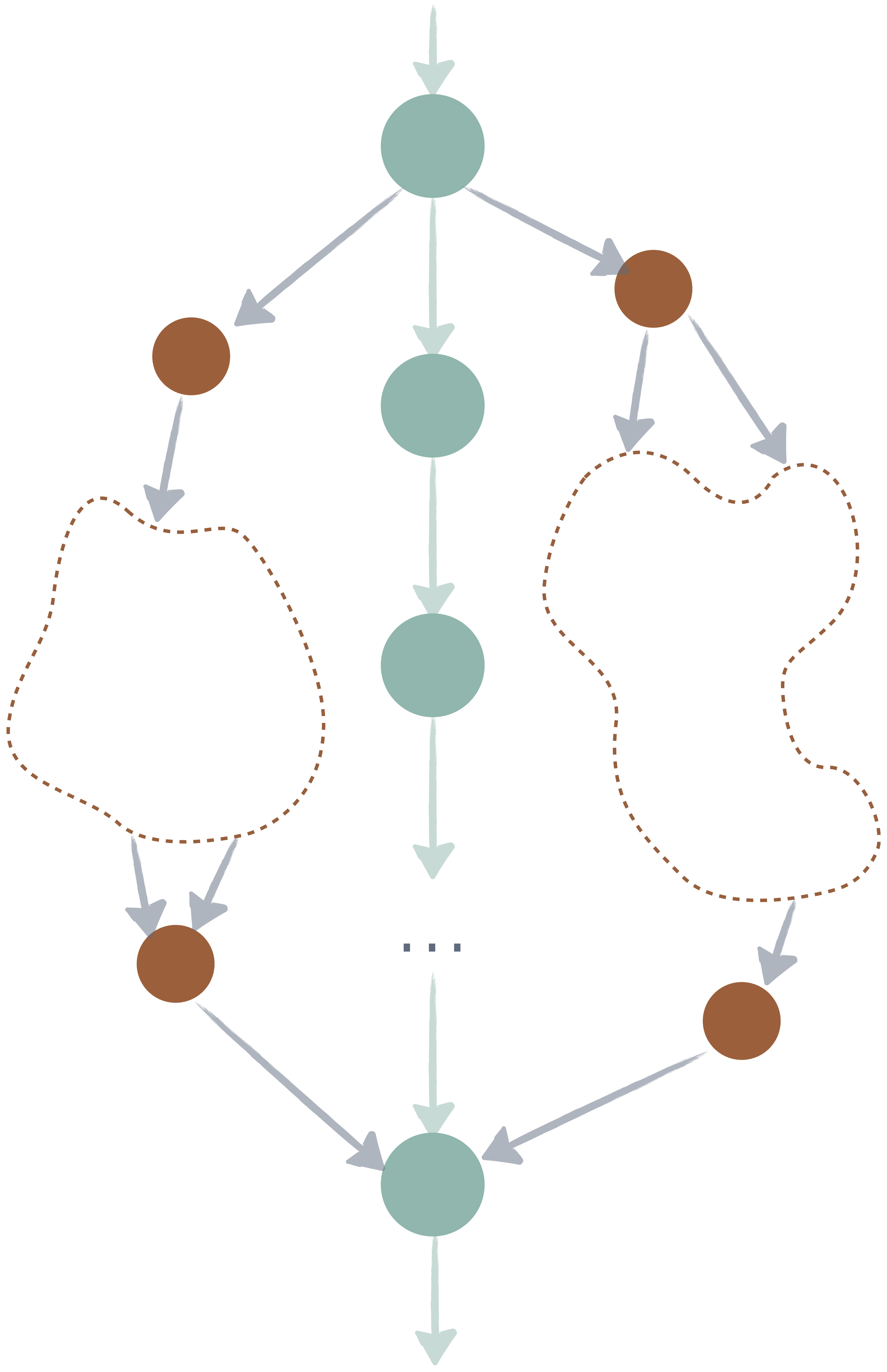


E.g., JàJà (1992), Blelloch (1994), …



E.g., Agarwal and Vitter (1988), …

Example: **Work-span** (depth) model



$W(n)$ = *work* (total ops)

$D(n)$ = *span* (critical path)

$W(n) \ / \ D(n)$
= **inherent parallelism**

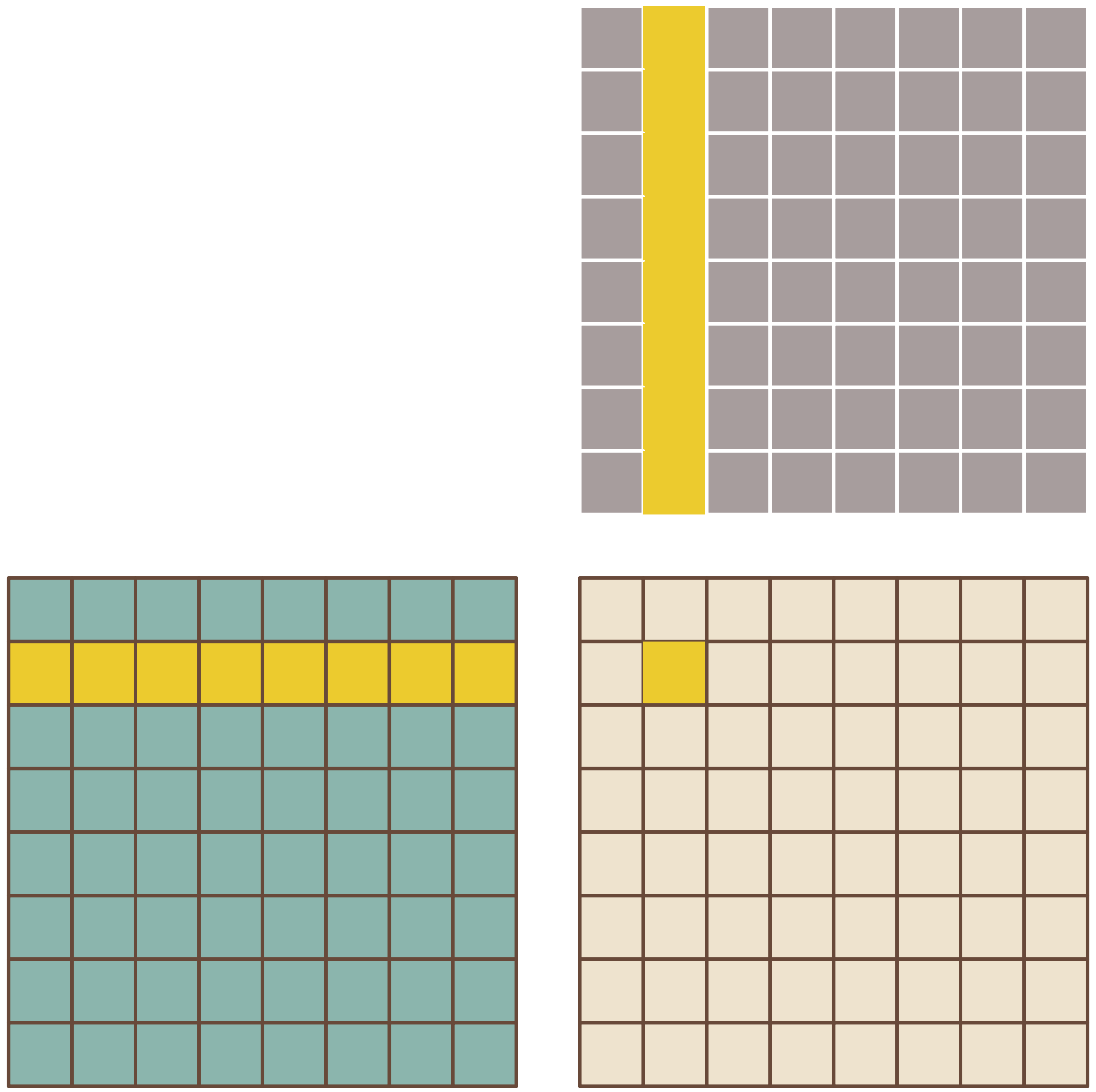
Desiderata:

Work optimality

Maximal parallelism

E.g., JàJà (1992), Blelloch (1994), …

Example: **Matrix multiply**
(non-Strassen)



$$C \leftarrow C + A * B$$

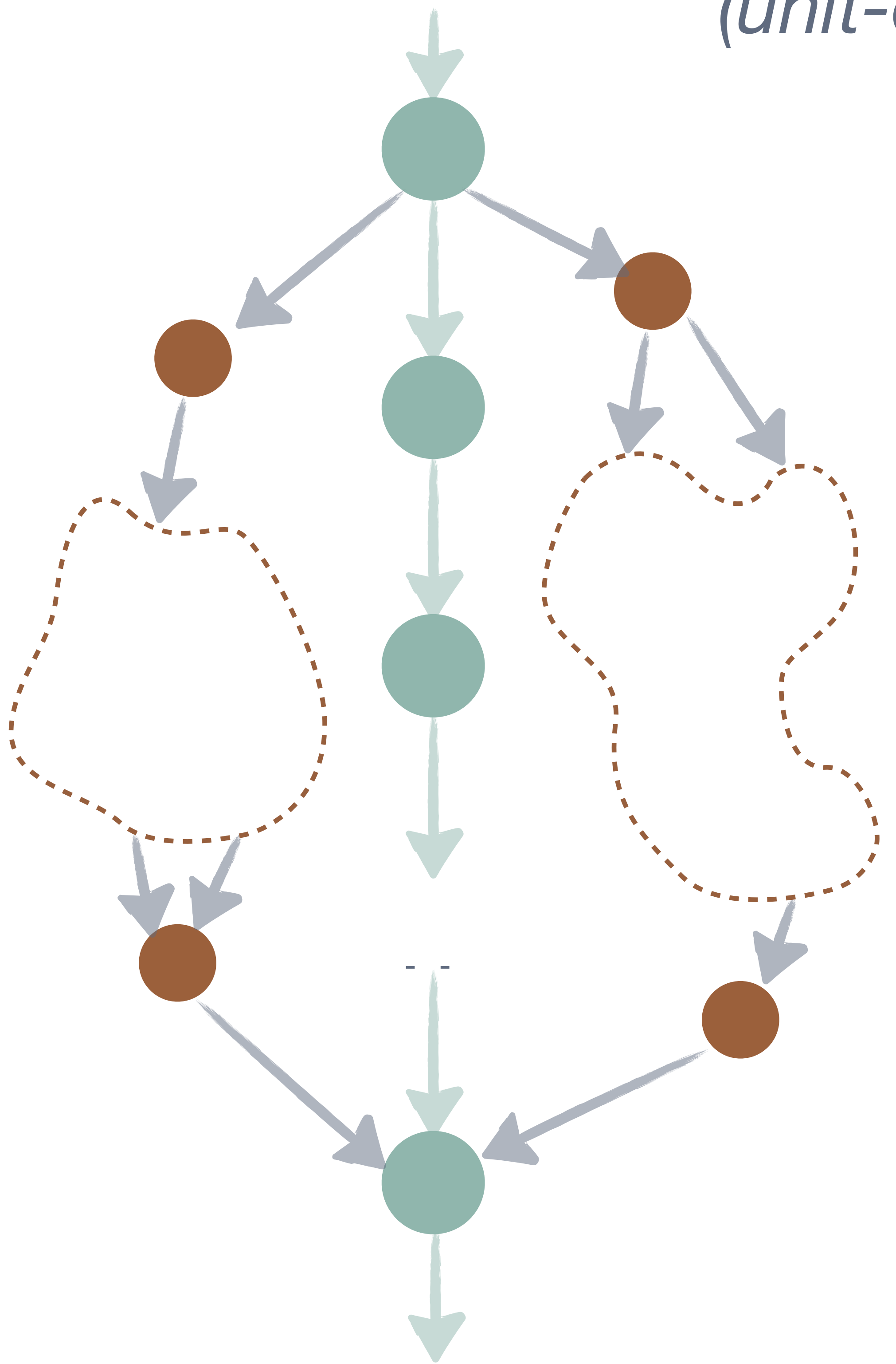
$$\frac{W(n)}{D(n)} = \Omega\left(\frac{n^3}{\log^k n}\right)$$

Parallelism

II. Moving from abstract to concrete (physical) costs

How much time to execute a DAG on P processors?

(unit-cost operations)



$W(n)$ = *work* (total ops)

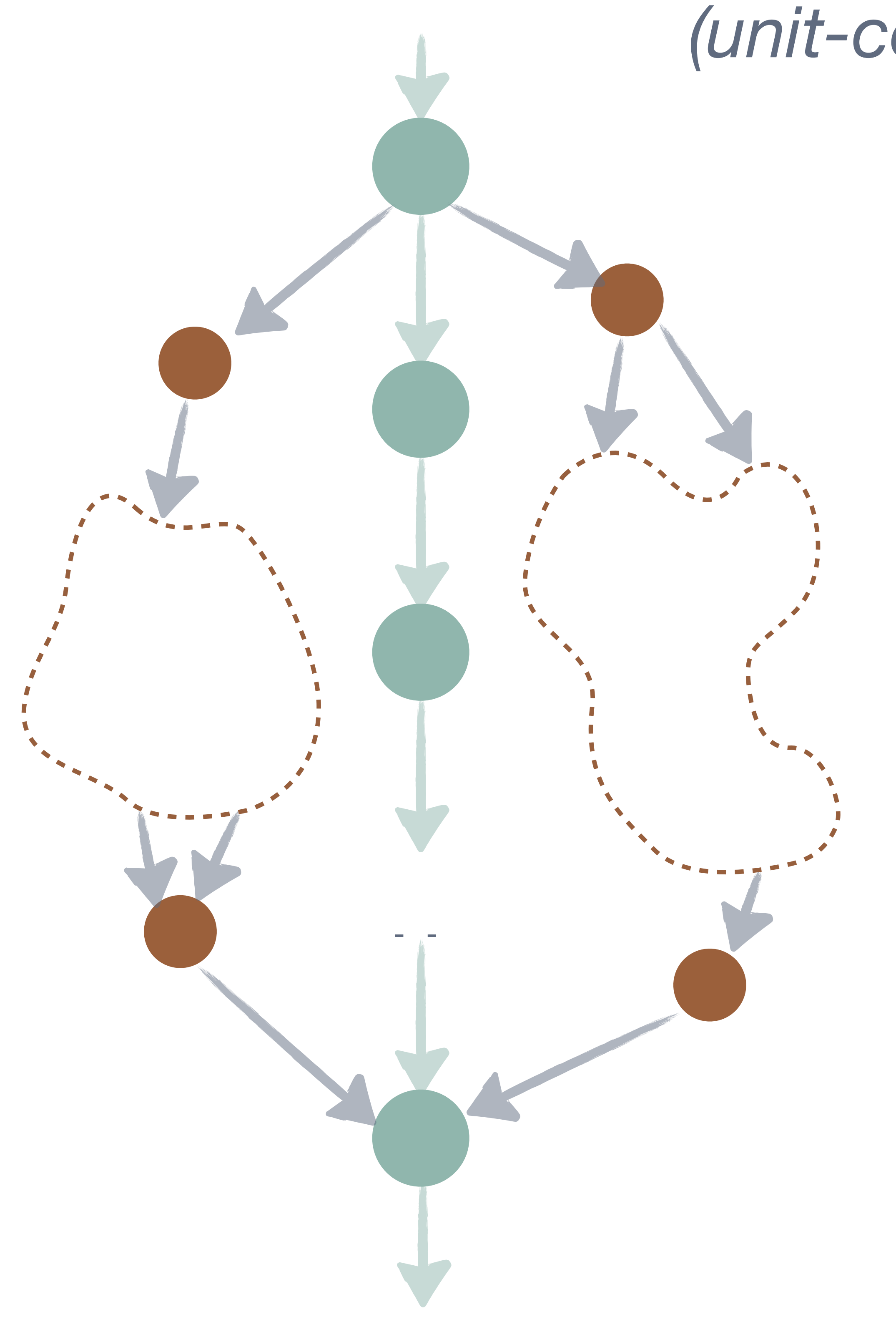
$D(n)$ = *span* (critical path)

$W(n)$ / $D(n)$
= ***inherent parallelism***

$$T(n; P) \quad = \quad ?$$

How much time to execute a DAG on P processors?

(unit-cost operations)



$W(n)$ = *work* (total ops)

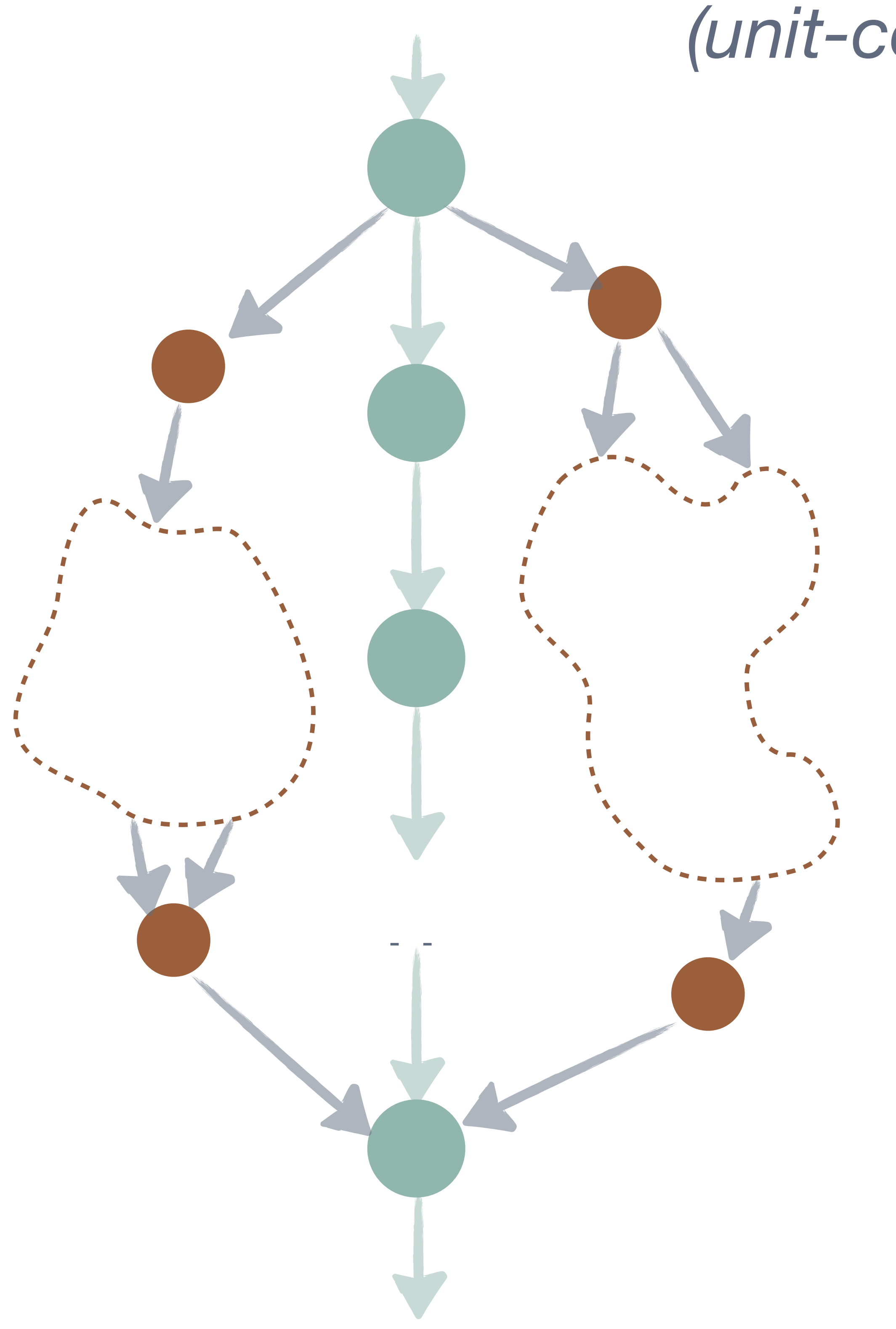
$D(n)$ = *span* (critical path)

$W(n)\ /\ D(n)$
= ***inherent parallelism***

$$T(n;P) \quad \geq \quad \max\left\{\frac{W(n)}{P}, D(n)\right\}$$

How much time to execute a DAG on P processors?

(unit-cost operations)



$W(n)$ = *work* (total ops)

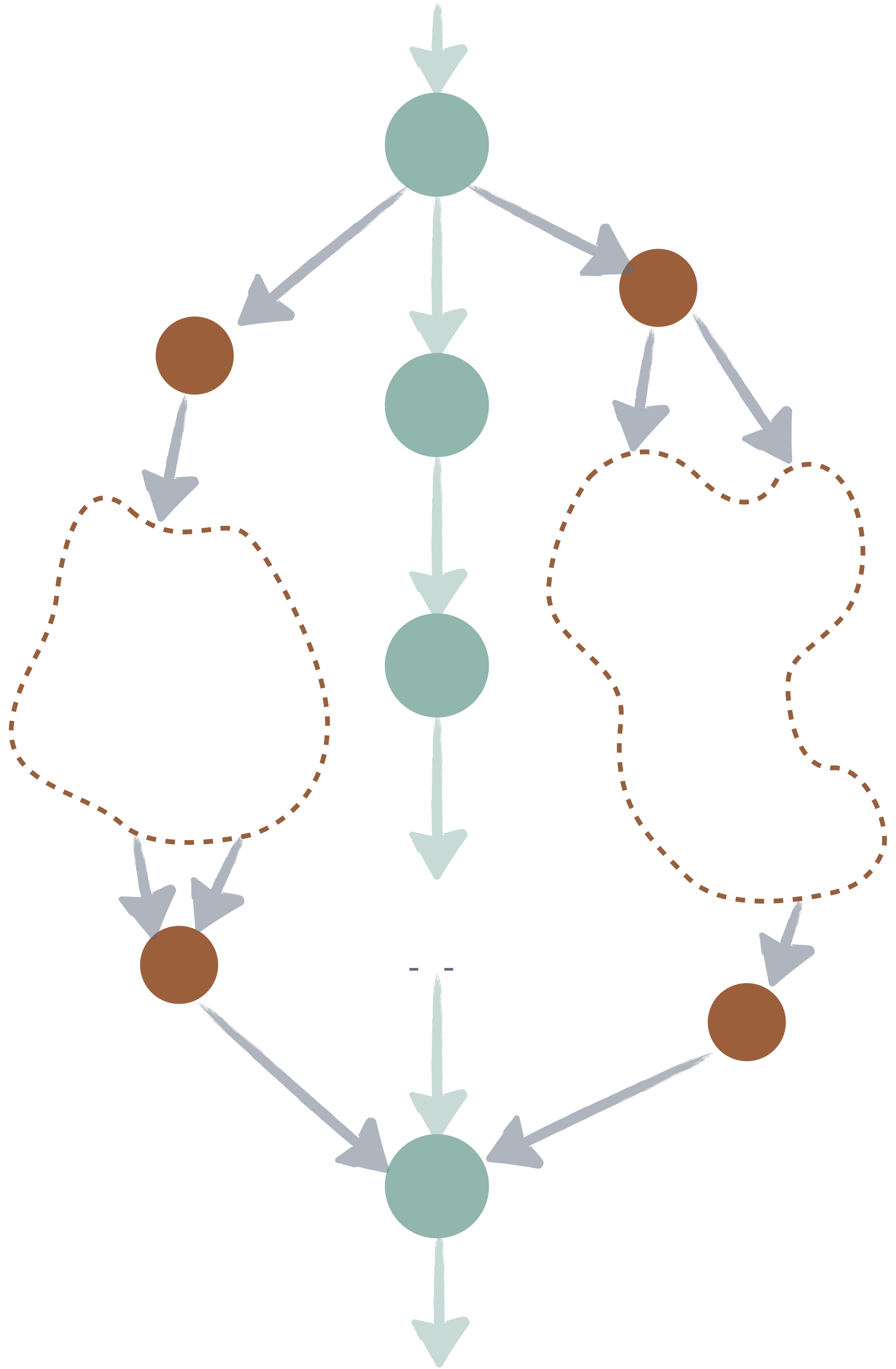
$D(n)$ = *span* (critical path)

$W(n)\ /\ D(n)$
= ***inherent parallelism***

$$T(n;P) \quad \le \quad D(n) + \frac{W(n) - D(n)}{P}$$

Brent (1975)

What is the speedup over the *best* sequential algorithm?



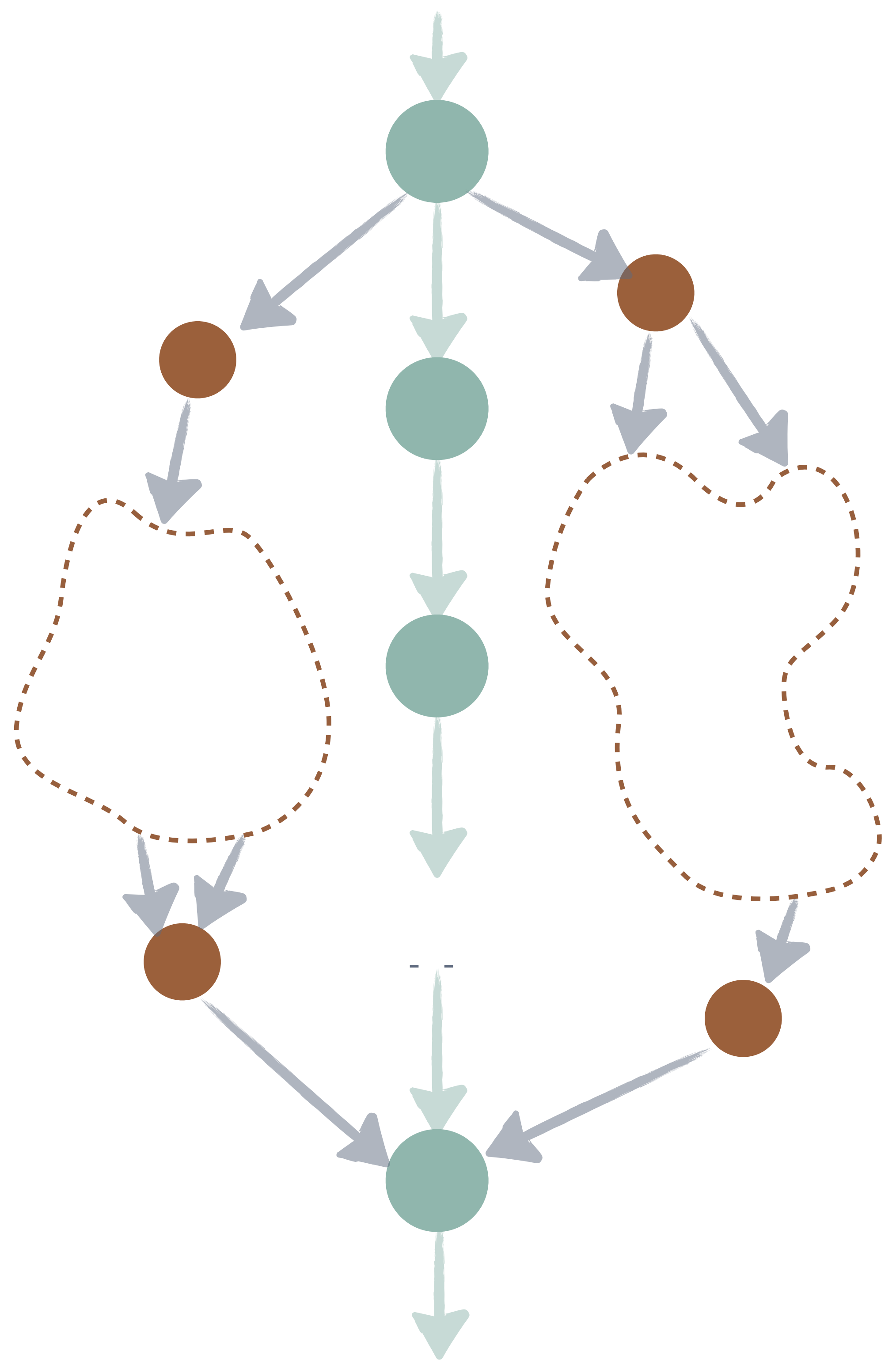
$W(n)$ = *work* (total ops)

$D(n)$ = *span* (critical path)

$W(n)\ /\ D(n)$
= ***inherent parallelism***

$$S_*(n;P) \equiv \frac{T_*(n)}{T(n;P)}$$

What is the speedup over the *best* sequential algorithm?



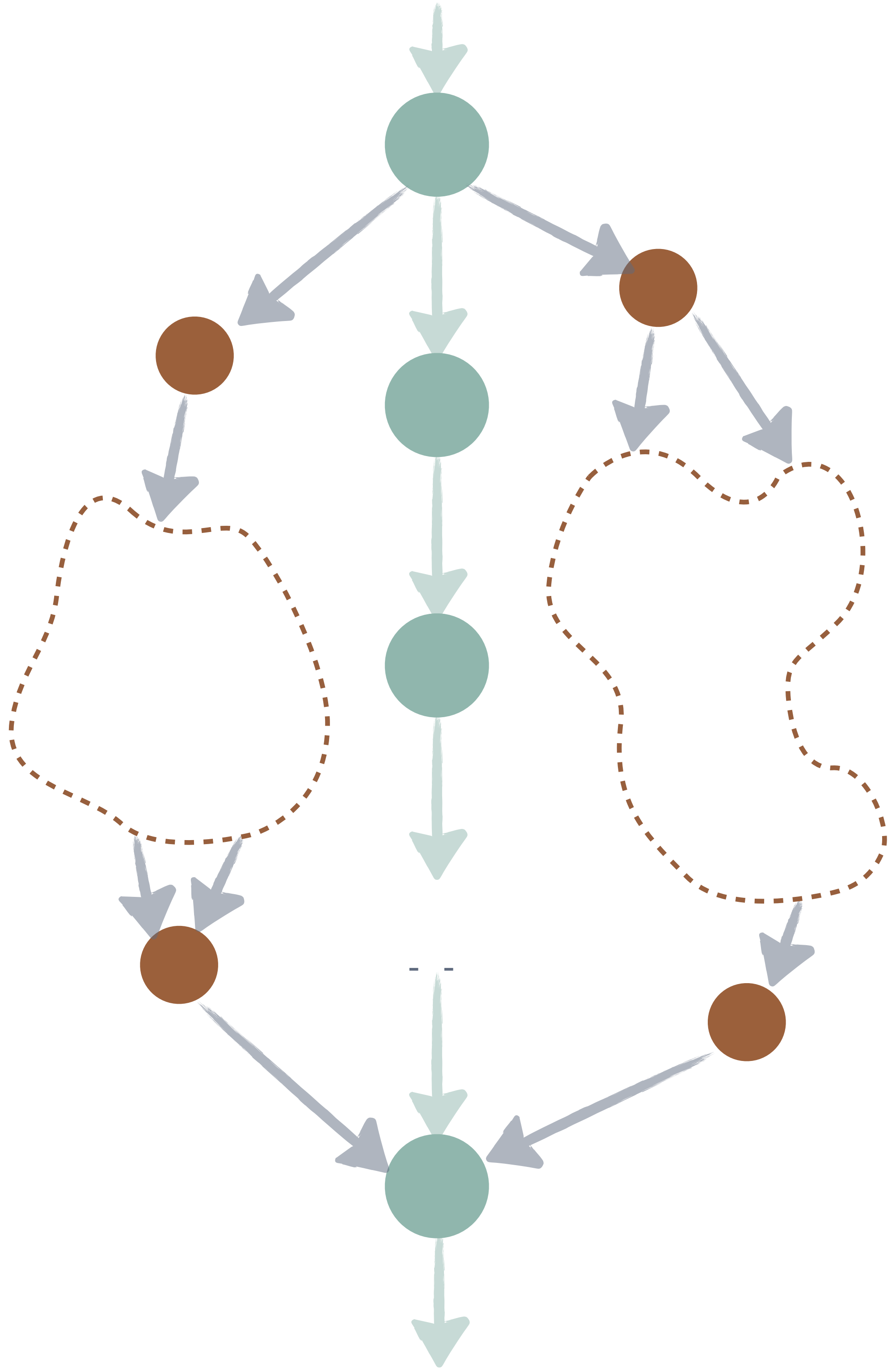
$W(n)$ = *work* (total ops)

$D(n)$ = *span* (critical path)

$W(n)\ /\ D(n)$
= ***inherent parallelism***

$$S_*(n;P) \quad \leq \quad \frac{W_*(n)}{\max\left\{\frac{W(n)}{P}, D(n)\right\}}$$

What is the speedup over the *best* sequential algorithm?



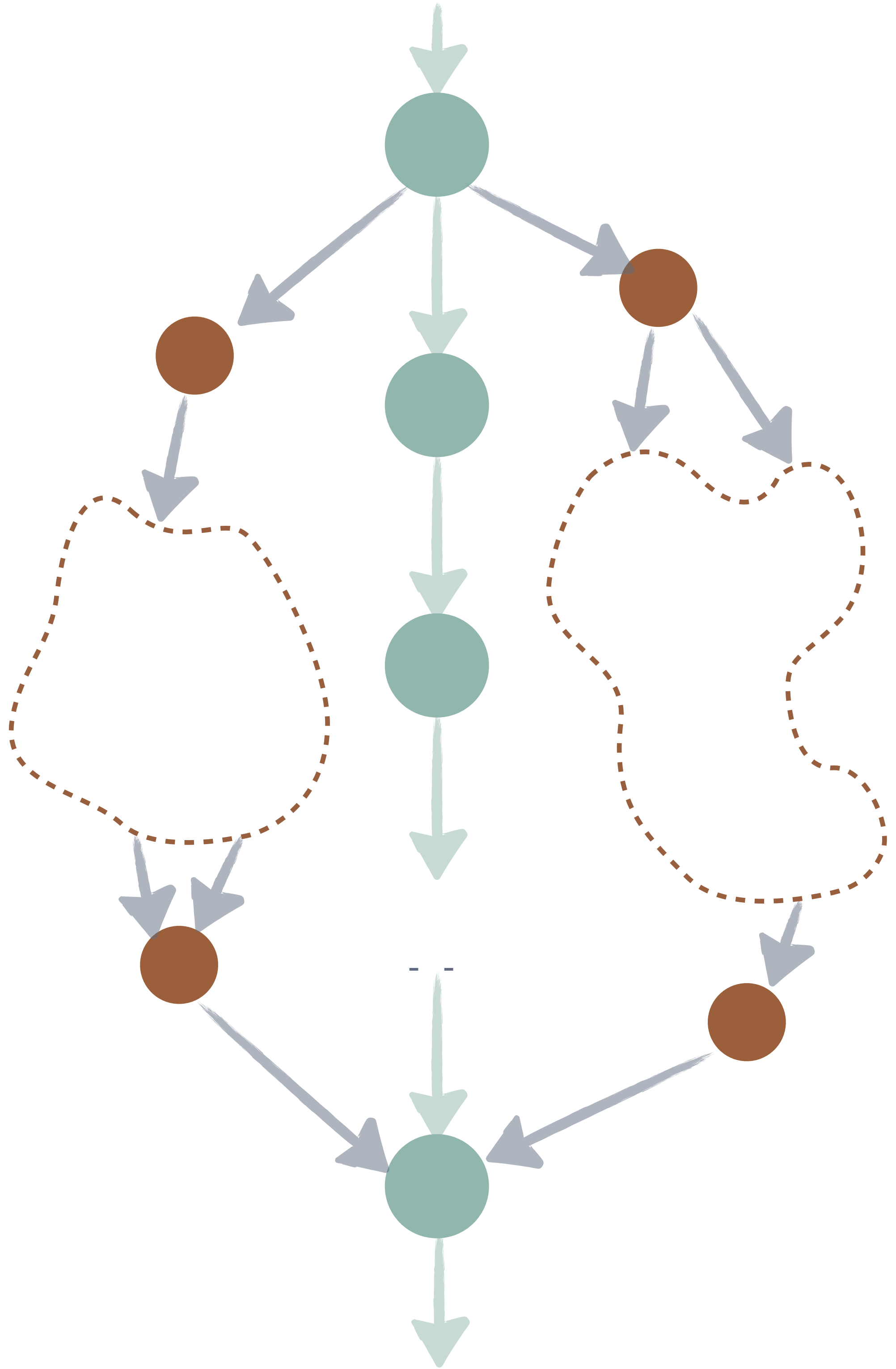
$W(n)$ = *work* (total ops)

$D(n)$ = *span* (critical path)

$W(n) \ / \ D(n)$
= ***inherent parallelism***

$$S_*(n;P) \leq P \cdot \min \left\{ \frac{W_*(n)}{W(n)}, \frac{W_*(n)/D(n)}{P} \right\}$$

What is the speedup over the *best* sequential algorithm?



$W(n)$ = *work* (total ops)

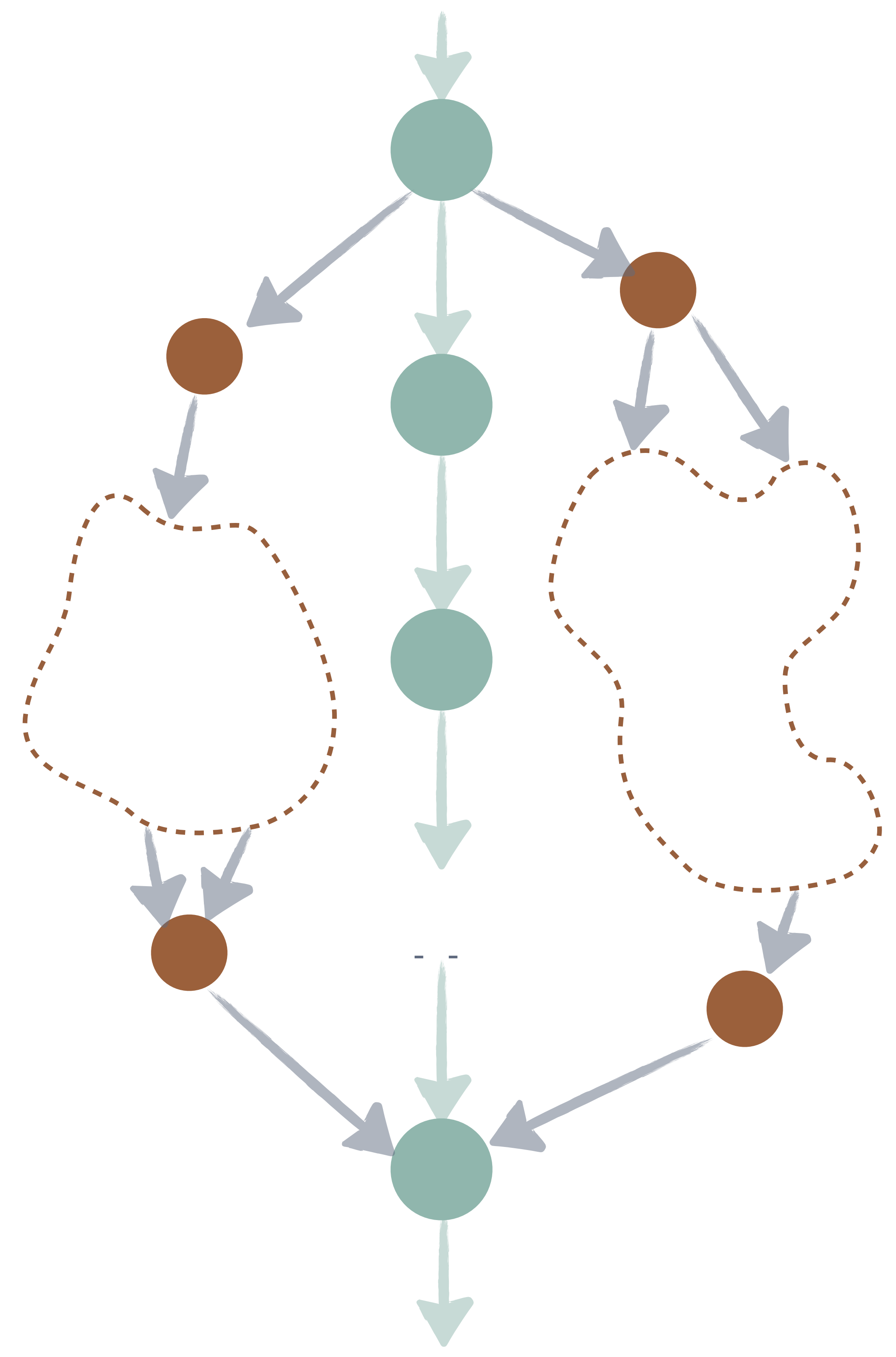
$D(n)$ = *span* (critical path)

$W(n)\ /\ D(n)$
= ***inherent parallelism***

Ideal (linear)

$$S_*(n;P) \leq P \cdot \min\left\{\frac{W_*(n)}{W(n)}, \frac{W_*(n)/D(n)}{P}\right\}$$

What is the speedup over the *best* sequential algorithm?



$W(n)$ = *work* (total ops)

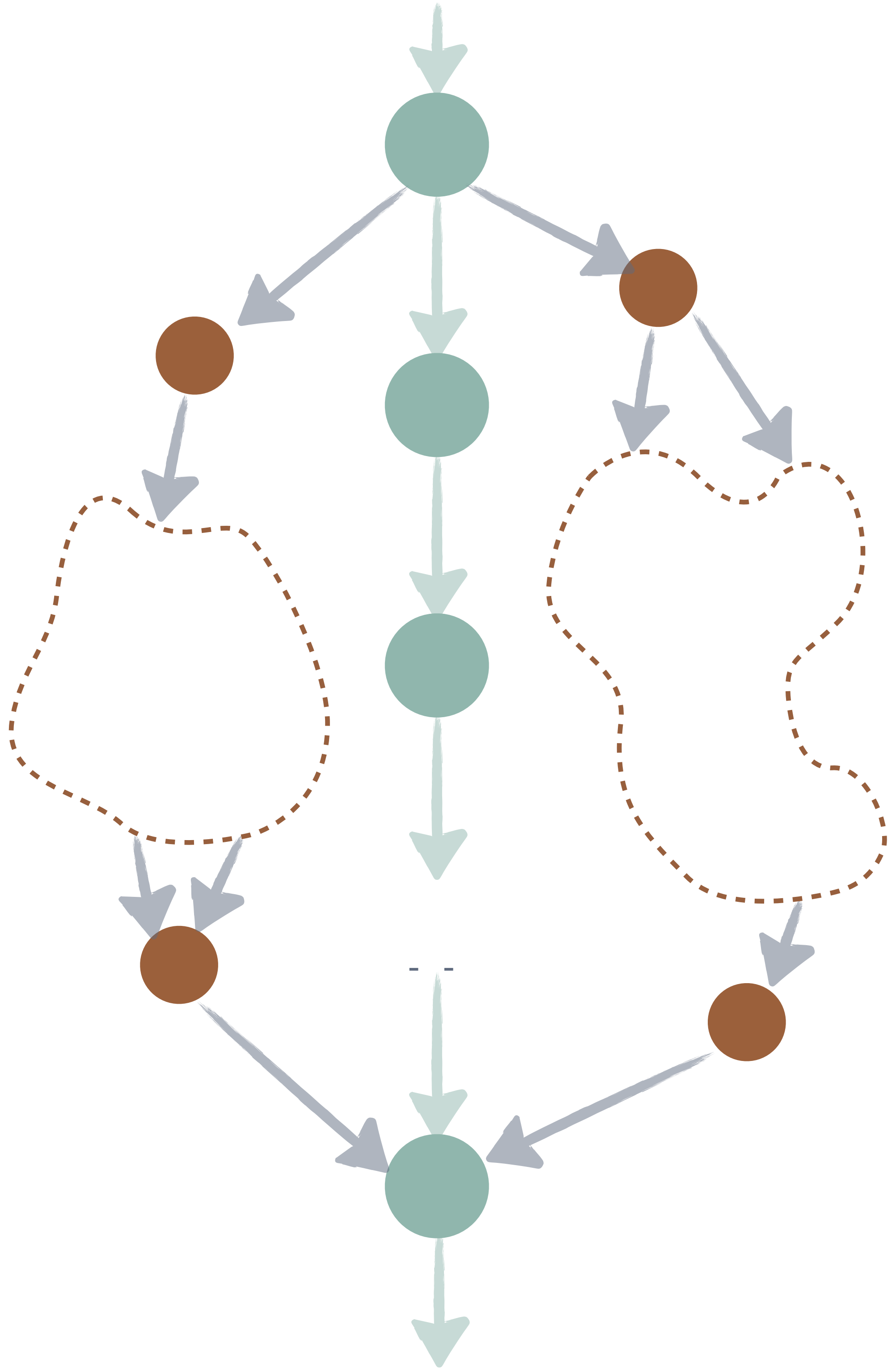
$D(n)$ = *span* (critical path)

$W(n)\ /\ D(n)$
= ***inherent parallelism***

Work-optimality

$$S_*(n;P) \leq P \cdot \min\left\{\frac{W_*(n)}{W(n)}, \frac{W_*(n)/D(n)}{P}\right\}$$

What is the speedup over the *best* sequential algorithm?



$W(n)$ = *work* (total ops)

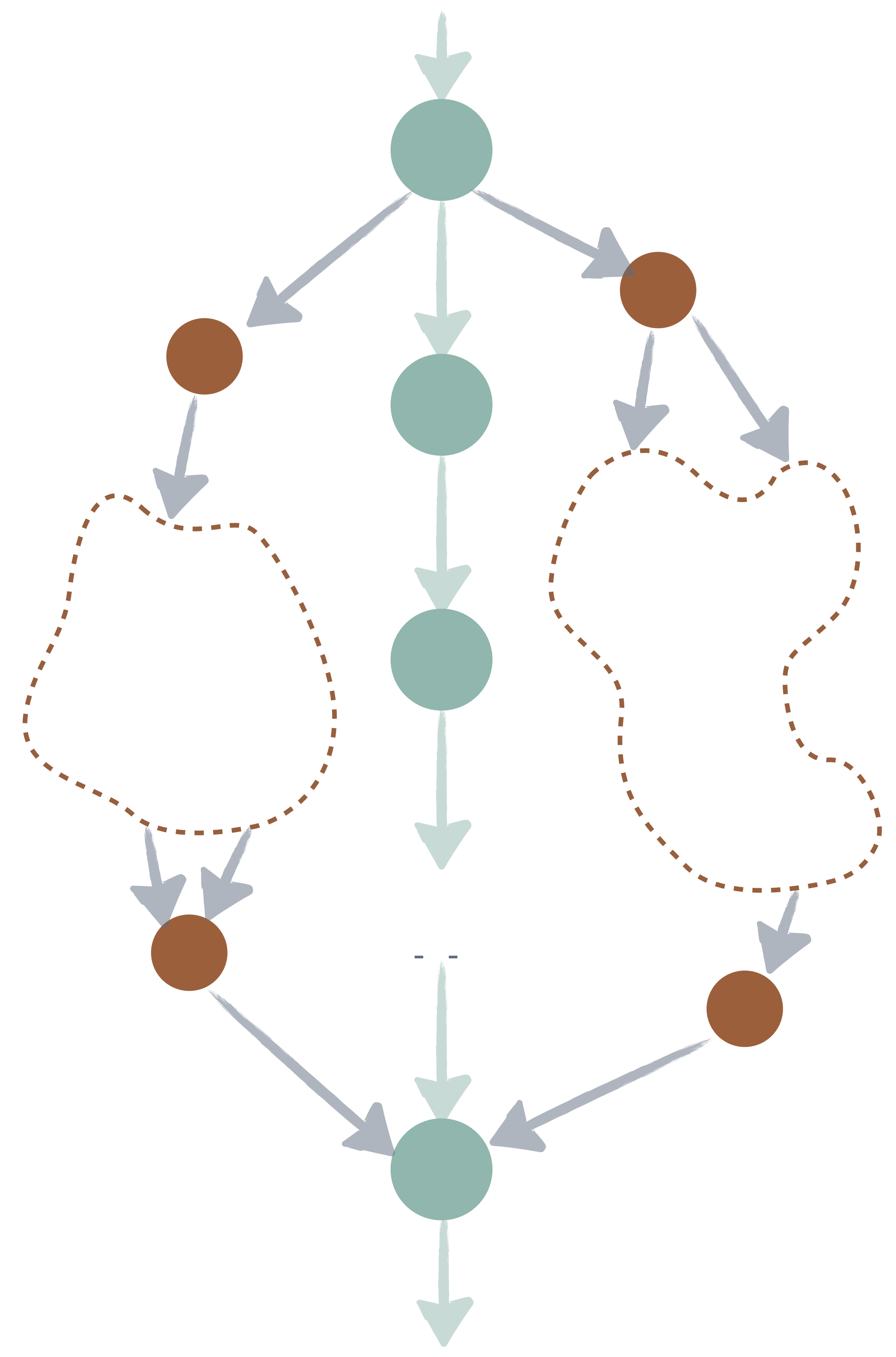
$D(n)$ = *span* (critical path)

$W(n) \;/\; D(n)$
= ***inherent parallelism***

Weak scalability

$$S_*(n;P) \leq P \cdot \min\left\{ \frac{W_*(n)}{W(n)}, \frac{W_*(n)/D(n)}{P} \right\}$$

What is the speedup over the *best* sequential algorithm?



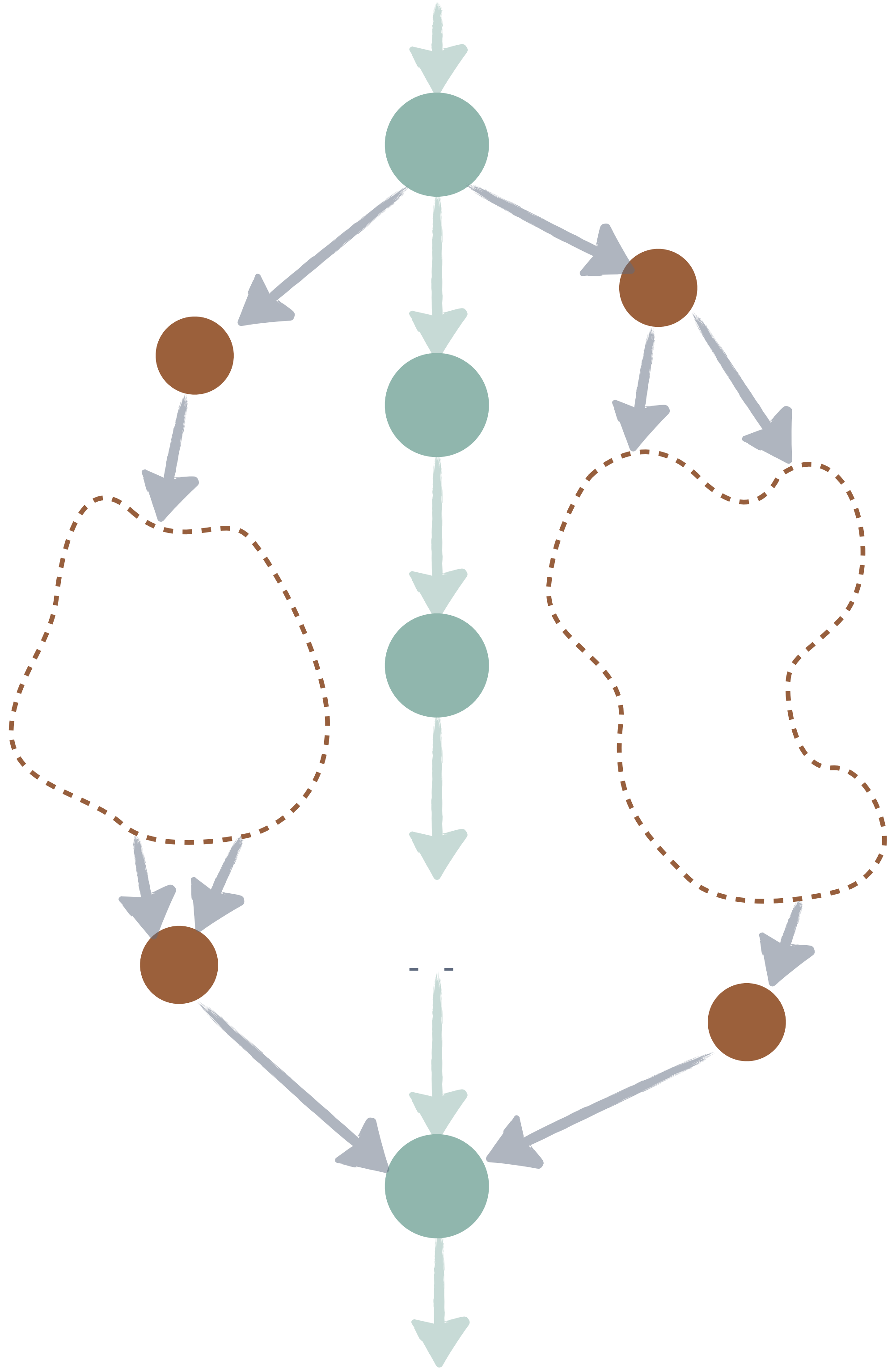
$W(n)$ = *work* (total ops)

$D(n)$ = *span* (critical path)

$W(n)\ /\ D(n)$
= ***inherent parallelism***

$$S_*(n;P) \geq \frac{W_*(n)}{D(n) + \frac{W(n)-D(n)}{P}}$$

What is the speedup over the *best* sequential algorithm?



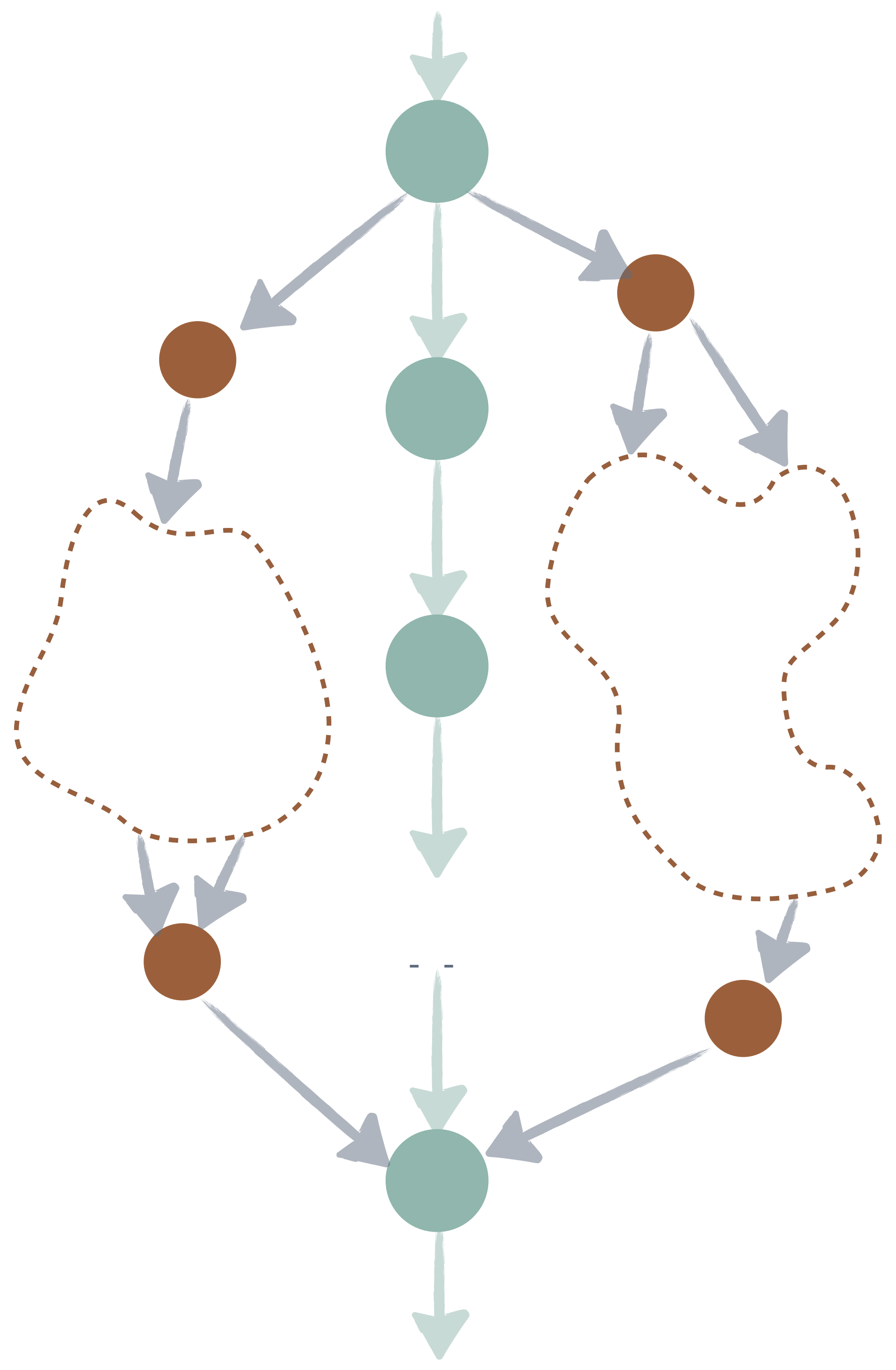
$W(n)$ = *work* (total ops)

$D(n)$ = *span* (critical path)

$W(n)\ /\ D(n)$
= ***inherent parallelism***

$$S_*(n;P) \quad \geq \quad \frac{P}{\frac{P-1}{W_*(n)/D(n)} + \frac{W(n)}{W_*(n)}}$$

How much **energy** is required?

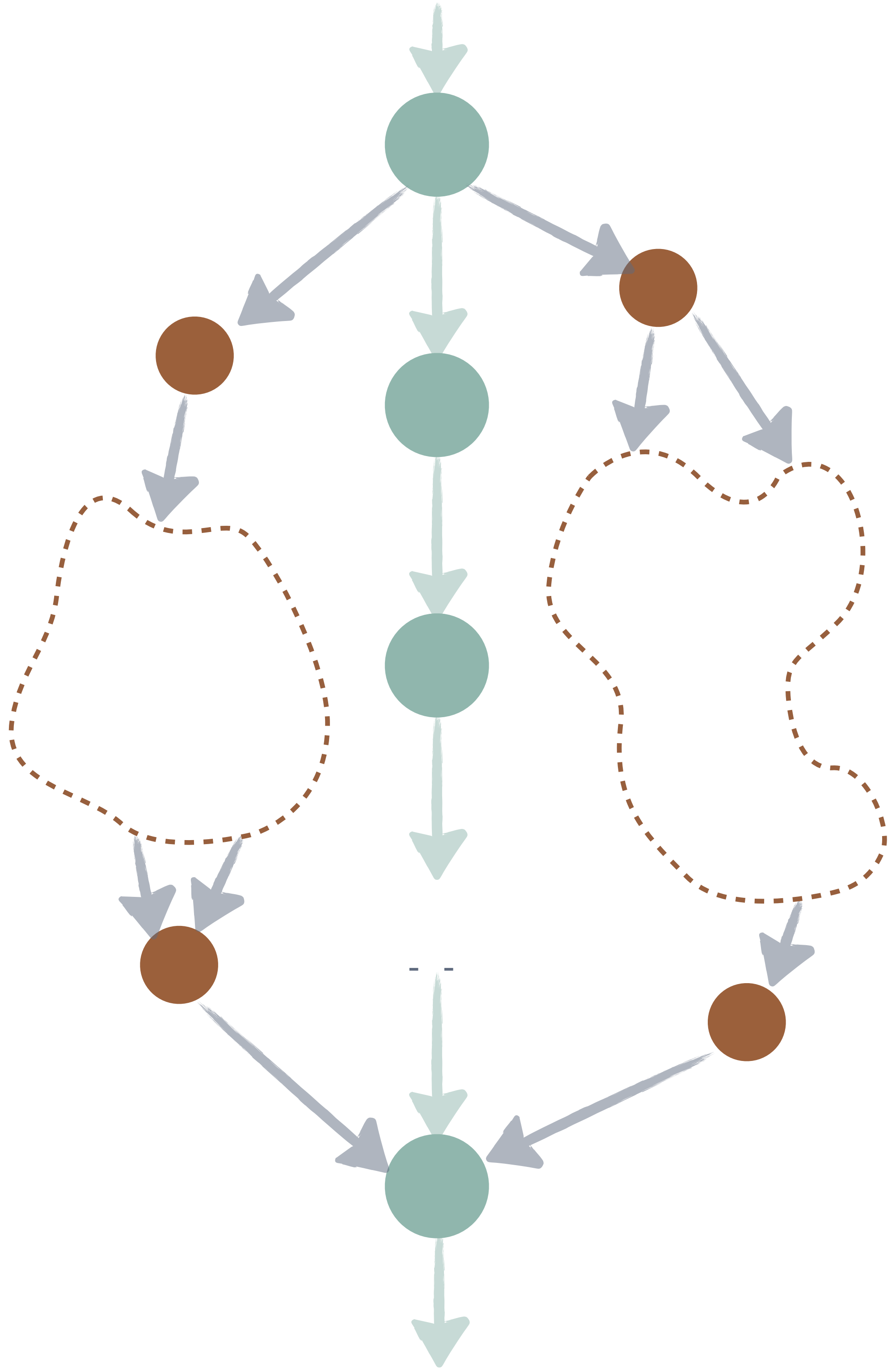


$W(n)$ = *work* (total ops)

$D(n)$ = *span* (critical path)

$W(n)\ /\ D(n)$
= ***inherent parallelism***

How much **energy** is required?



$W(n)$ = *work* (total ops)

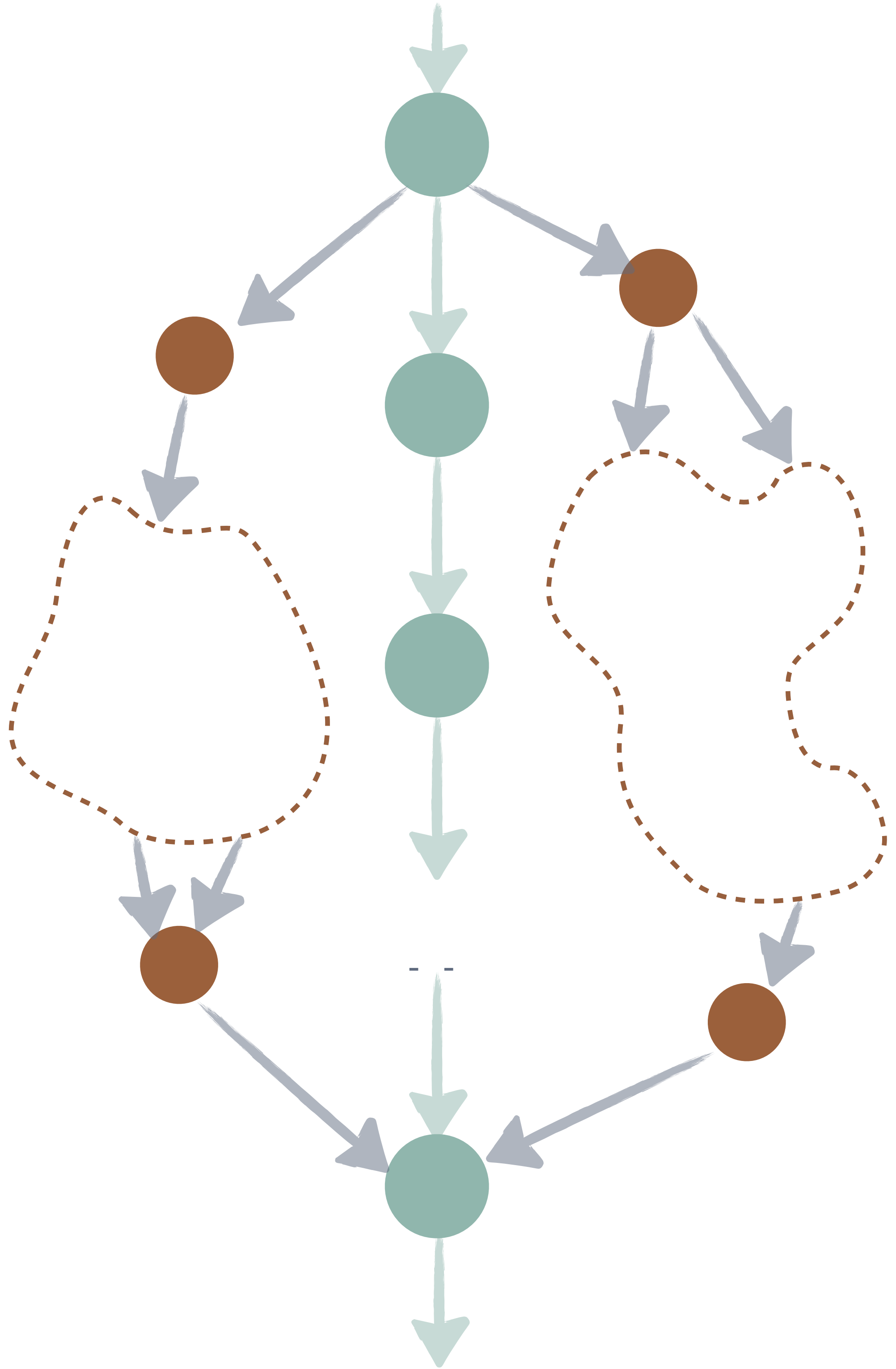
$D(n)$ = *span* (critical path)

$W(n)\ /\ D(n)$
= ***inherent parallelism***

Time is a cost you ***may hide*** by overlap, e.g., parallelism.

Energy is a cost you ***must pay*** for every operation.

How much **energy** is required?



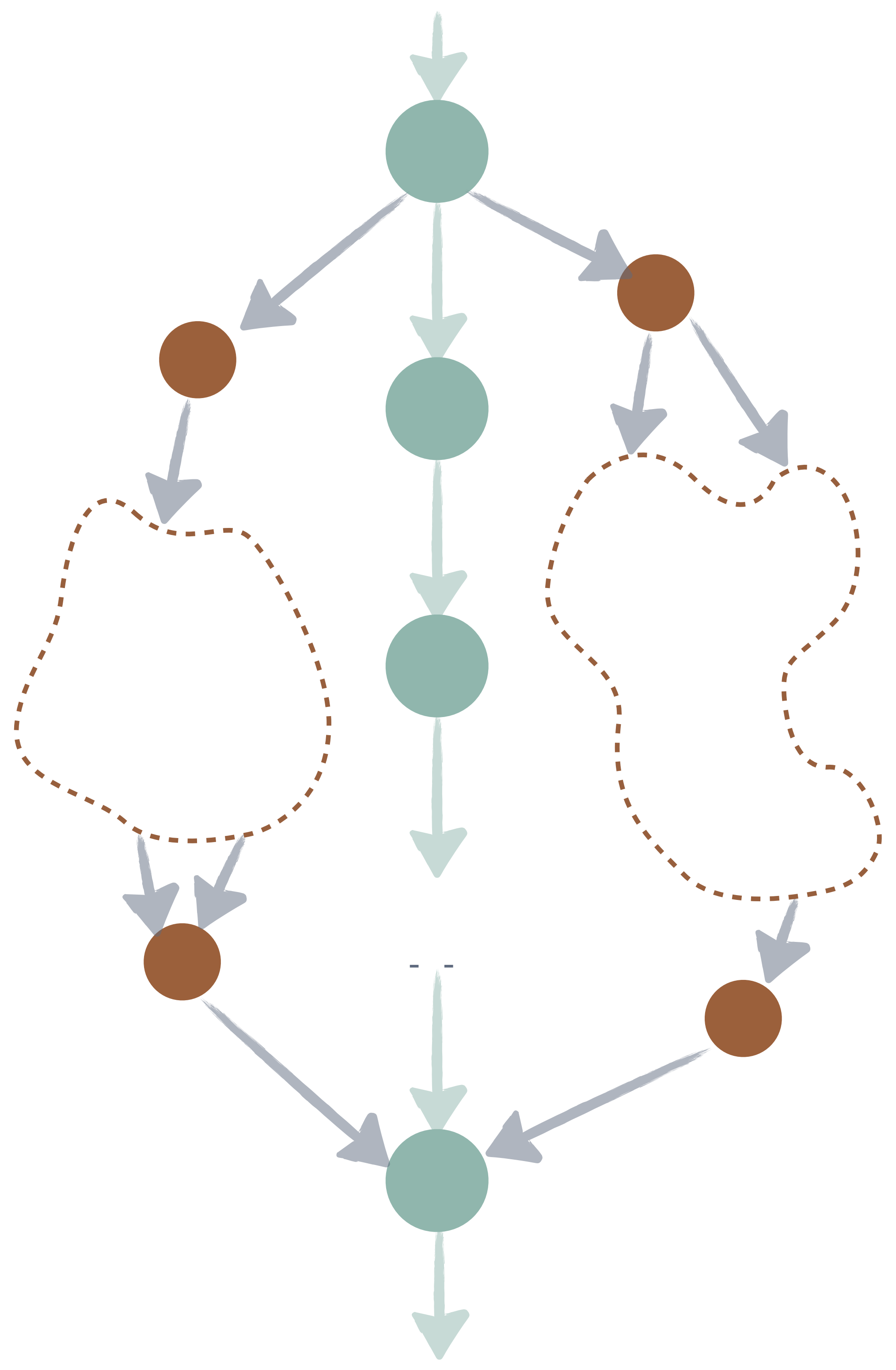
$W(n)$ = *work* (total ops)

$D(n)$ = *span* (critical path)

$W(n)\ /\ D(n)$
= ***inherent parallelism***

Energy $\propto$ **Work**

How much **energy** is required?



$W(n)$ = *work* (total ops)

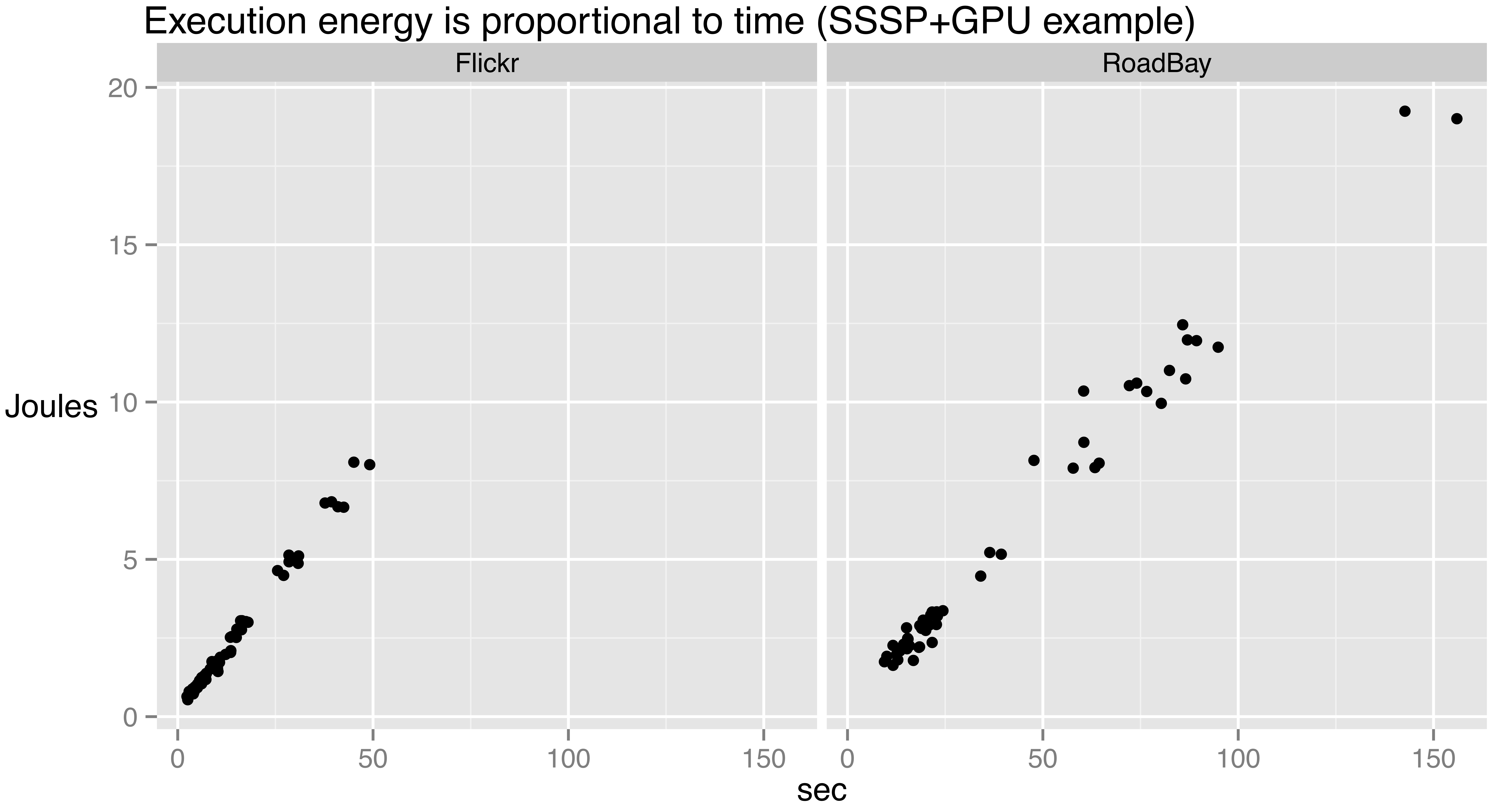
$D(n)$ = *span* (critical path)

$W(n)\ /\ D(n)$
= ***inherent parallelism***

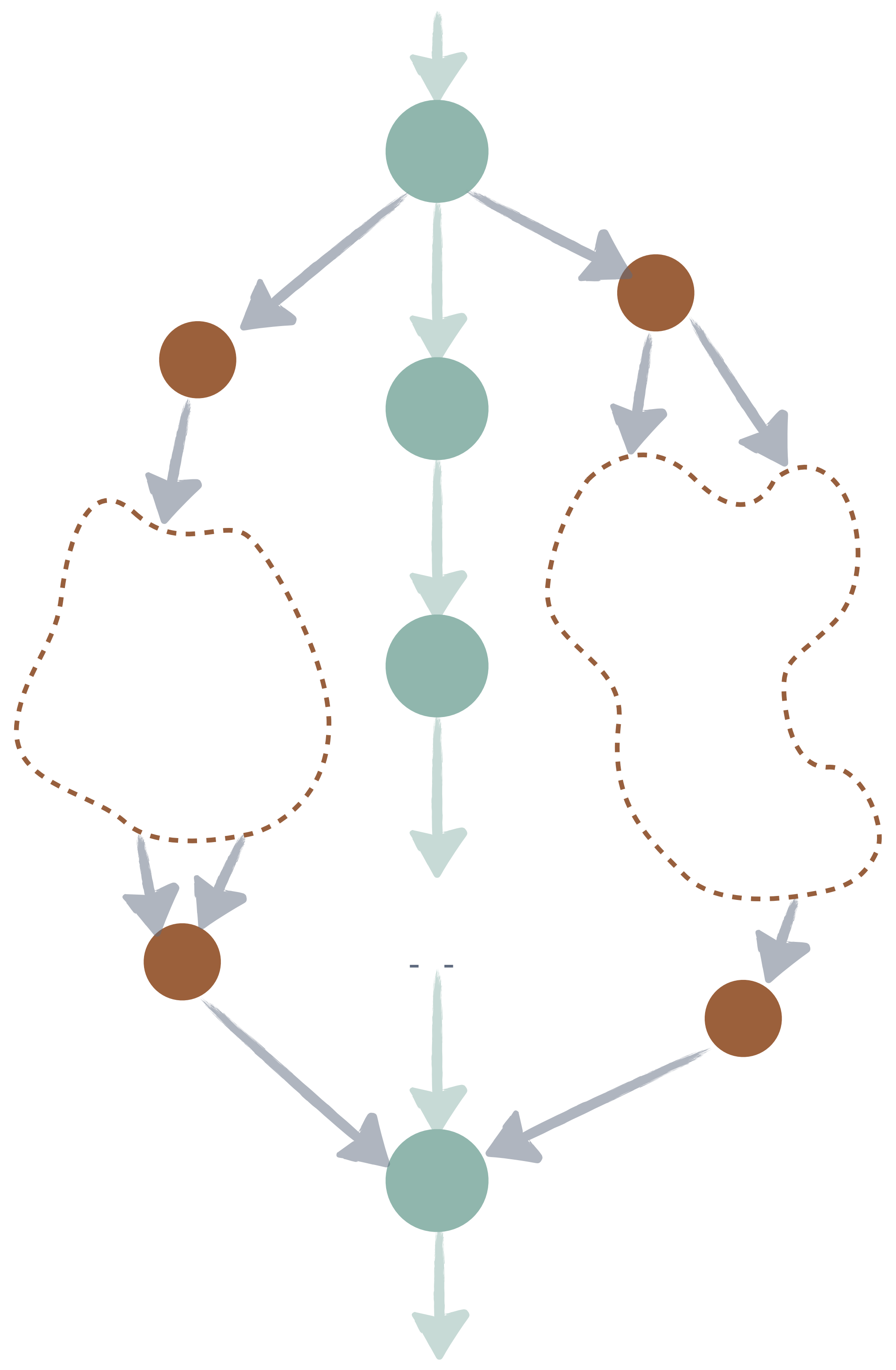
Energy $\propto$ **Work**

$$T(n;P) \sim \frac{W(n)}{P}$$

Time $\propto$ **Energy**


Execution energy is proportional to time (SSSP+GPU example)
Flickr
RoadBay
Joules
sec
0
5
10
15
20
0
50
100
150

How much **power** is required?

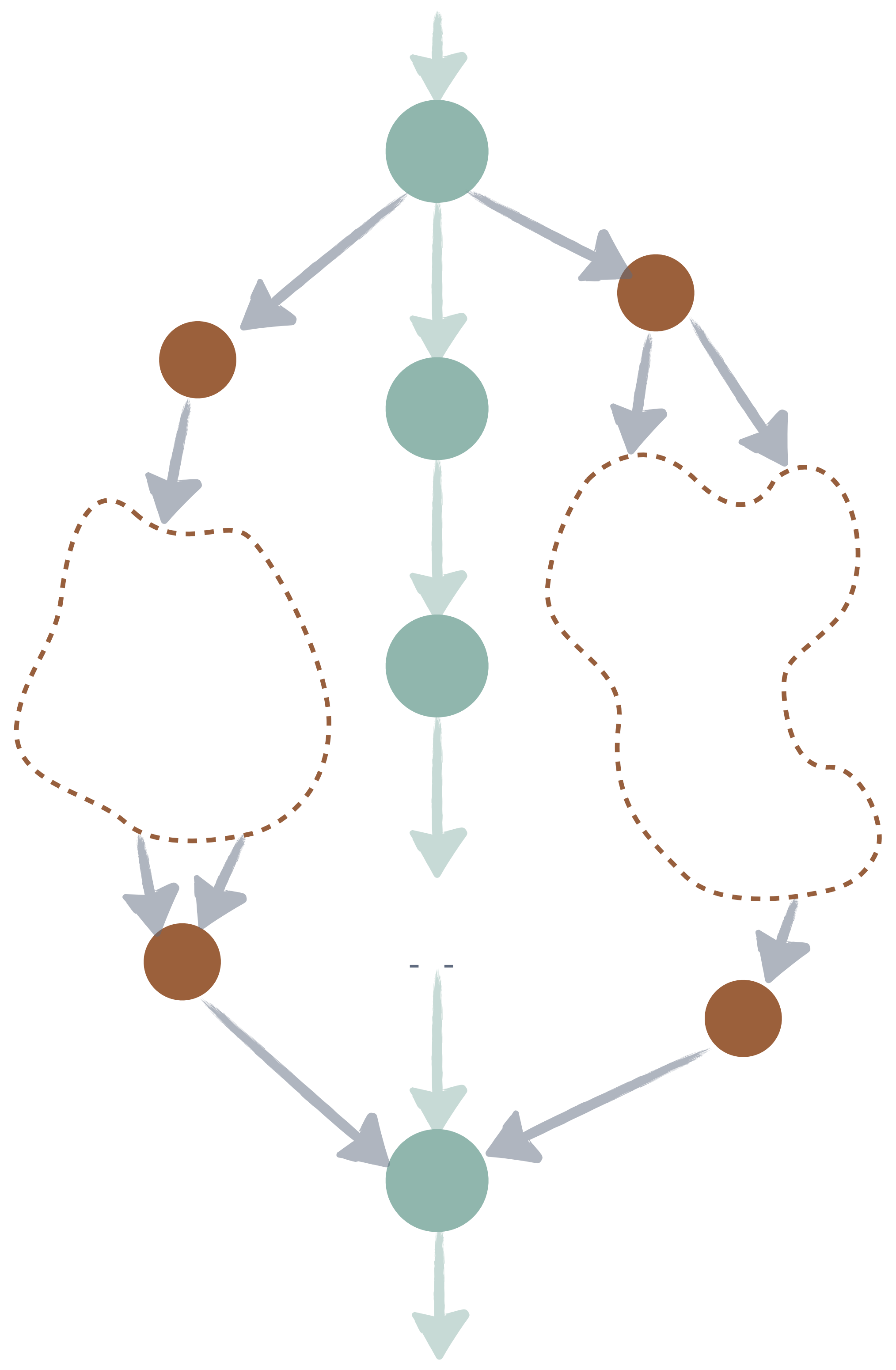


$W(n)$ = *work* (total ops)

$D(n)$ = *span* (critical path)

$W(n)\ /\ D(n)$
= ***inherent parallelism***

How much **power** is required?



$W(n)$ = *work* (total ops)

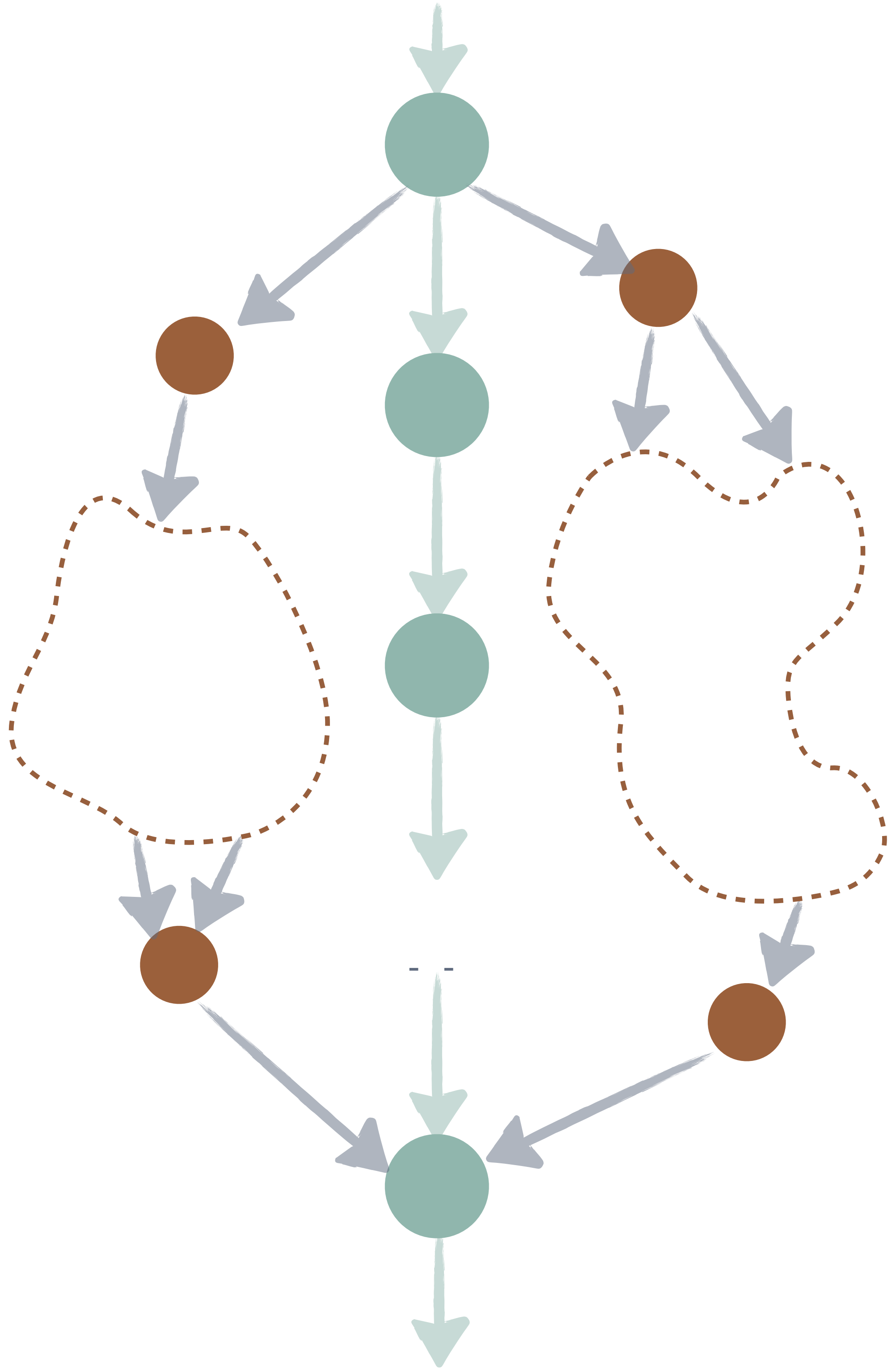
$D(n)$ = *span* (critical path)

$W(n) \;/\; D(n)$
= ***inherent parallelism***

Power ≡ Energy / Time

(average instantaneous)

How much **power** is required?



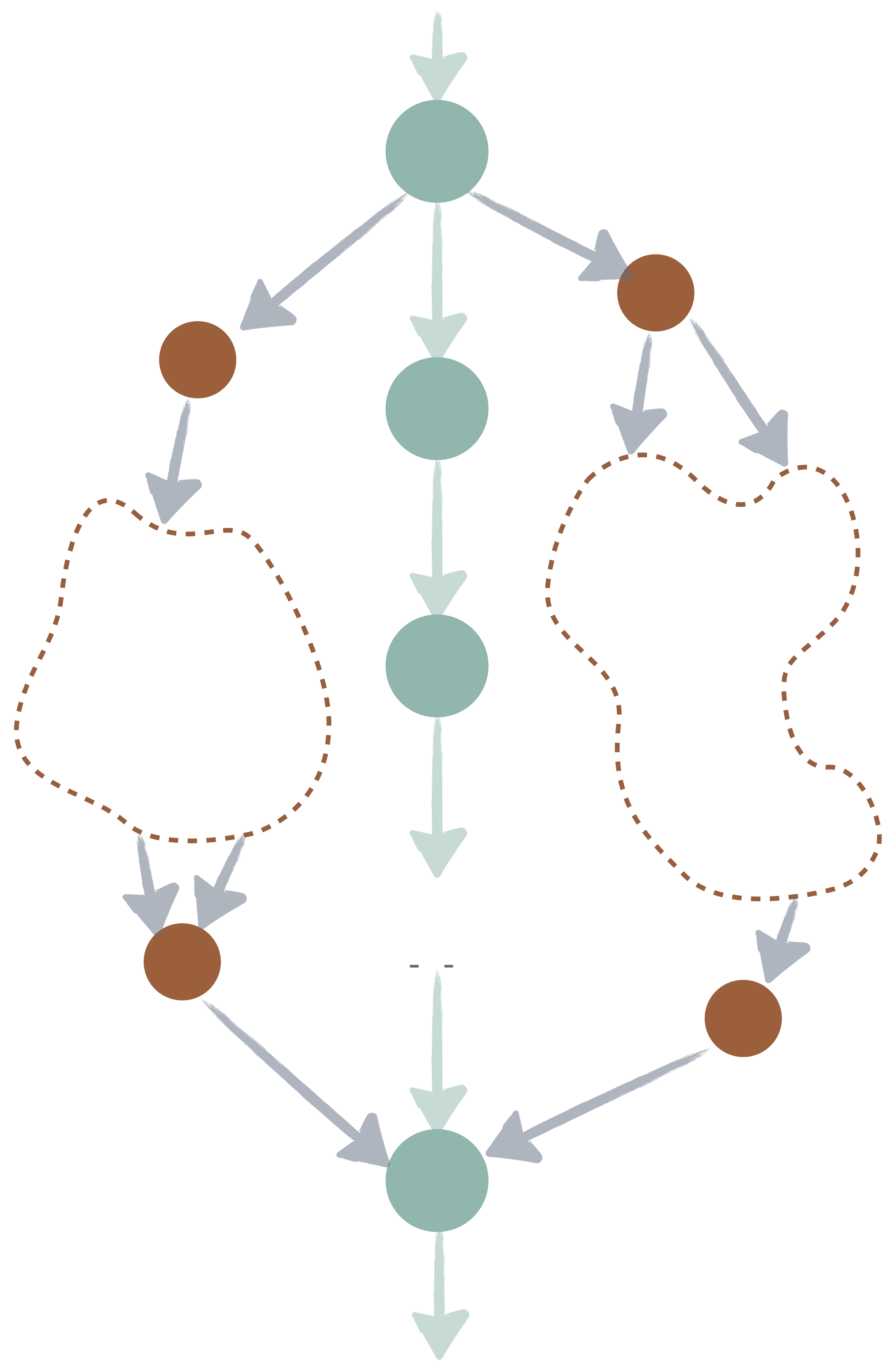
$W(n)$ = *work* (total ops)

$D(n)$ = *span* (critical path)

$W(n)\ /\ D(n)$
= ***inherent parallelism***

$$\Phi(n;P) \equiv \frac{\Theta(W(n))}{T(n;P)}$$

How much **power** is required?



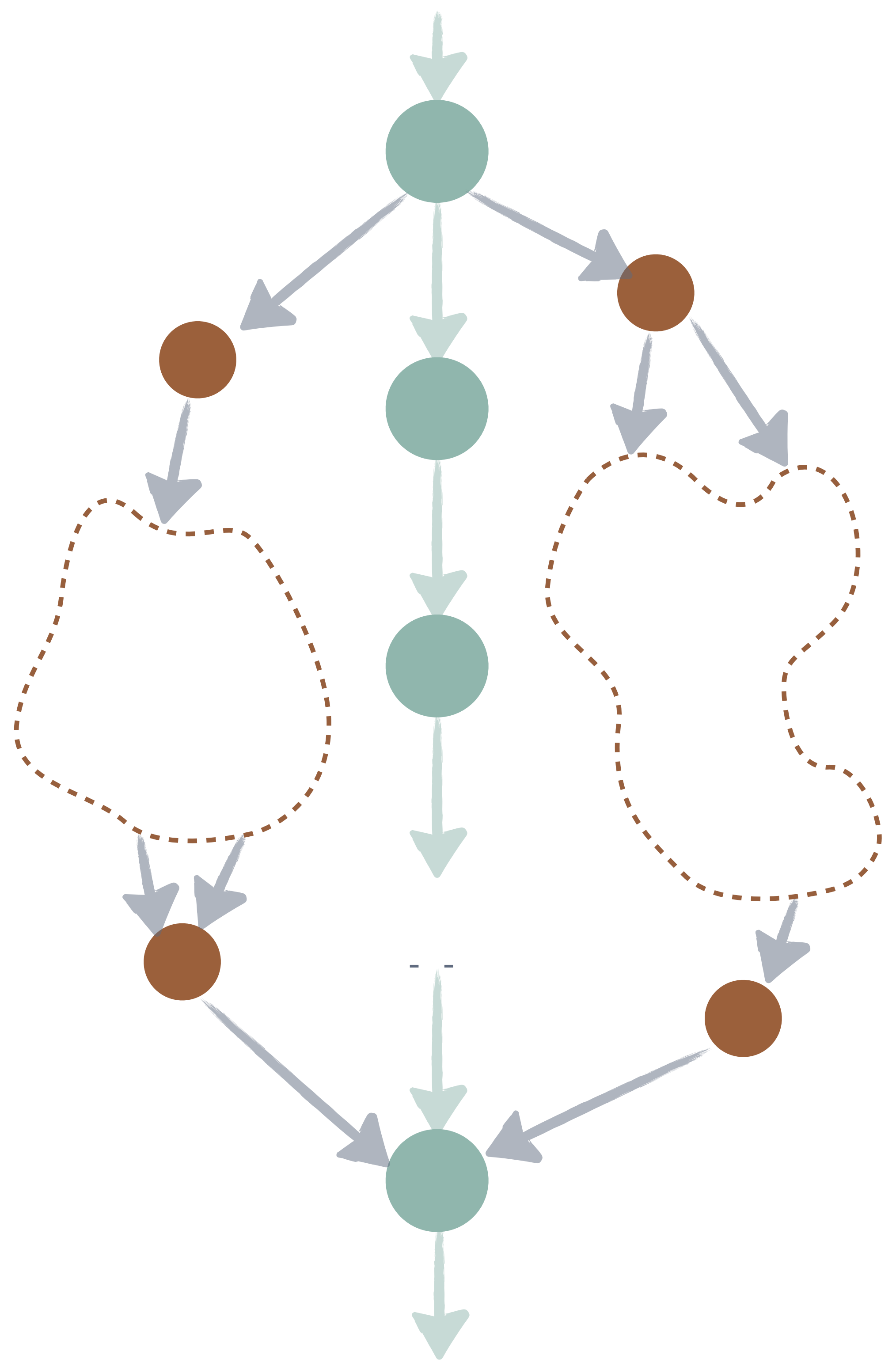
$W(n)$ = *work* (total ops)

$D(n)$ = *span* (critical path)

$W(n)\ /\ D(n)$
= ***inherent parallelism***

$$\Phi(n;P) = \frac{\Theta\left(T(n;P=1)\right)}{T(n;P)} = \Theta\left(S(n;P)\right)$$

How much **power** is required?



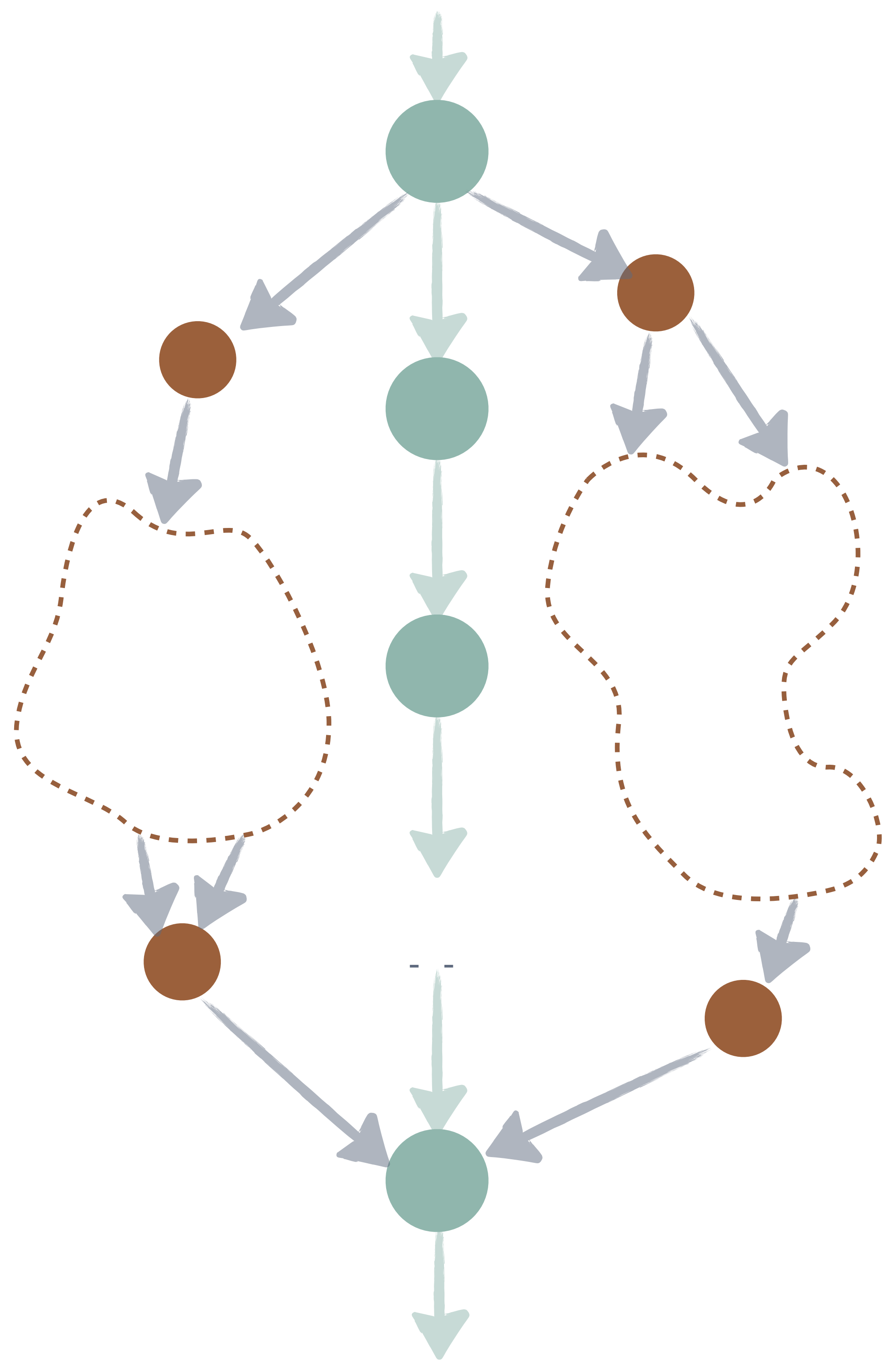
$W(n)$ = *work* (total ops)

$D(n)$ = *span* (critical path)

$W(n)$ / $D(n)$
= ***inherent parallelism***

$$\Phi(n;P) = \frac{\Theta\left(T(n;P=1)\right)}{T(n;P)} = \Theta\left(S(n;P)\right)$$

How much **power** is required?



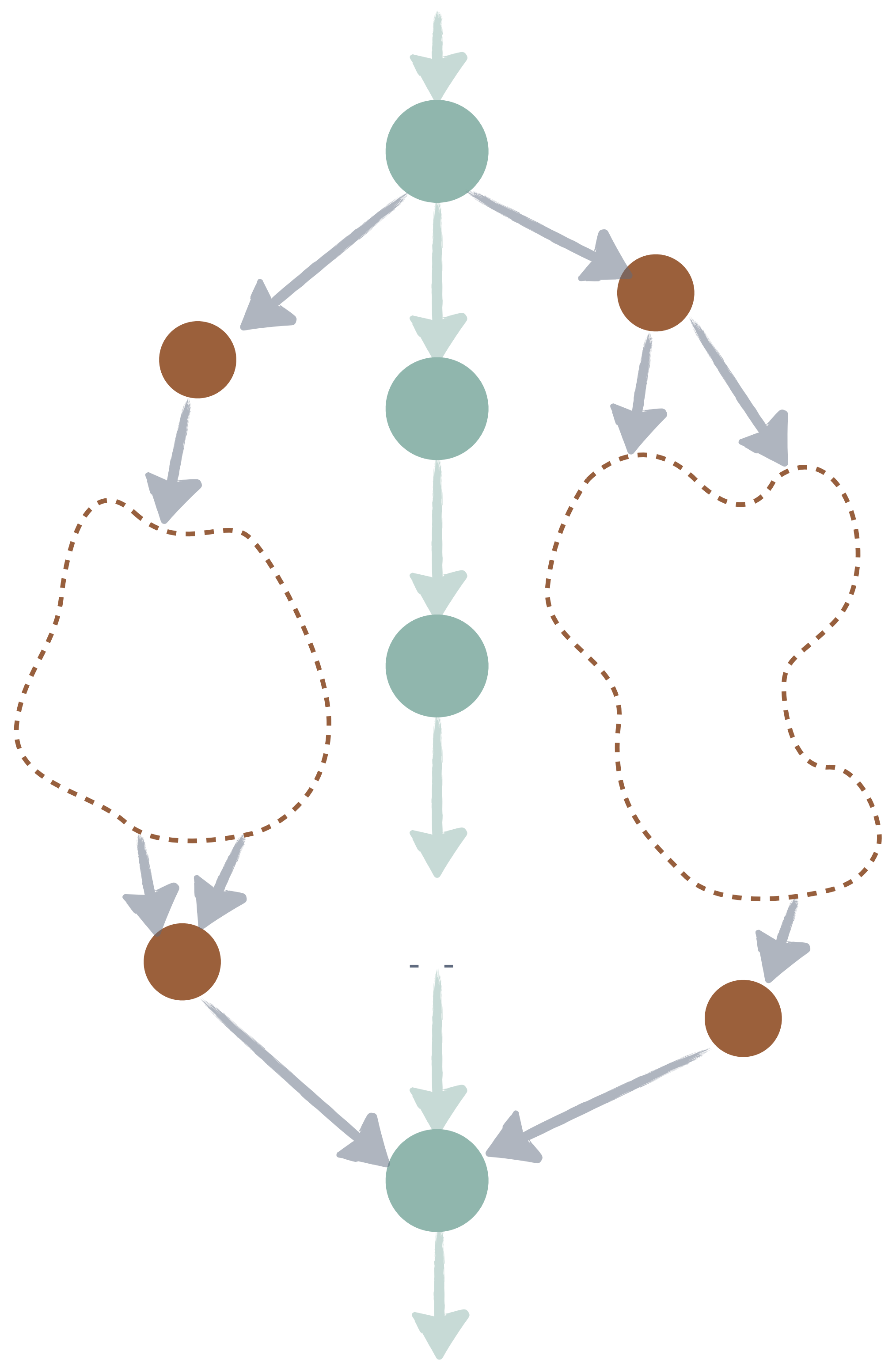
$W(n)$ = *work* (total ops)

$D(n)$ = *span* (critical path)

$W(n)$ / $D(n)$
= ***inherent parallelism***

$$\Phi(n;P) = \frac{\Theta\left(T(n;P=1)\right)}{T(n;P)} = \Theta\left(S(n;P)\right)$$

How much **power** is required?

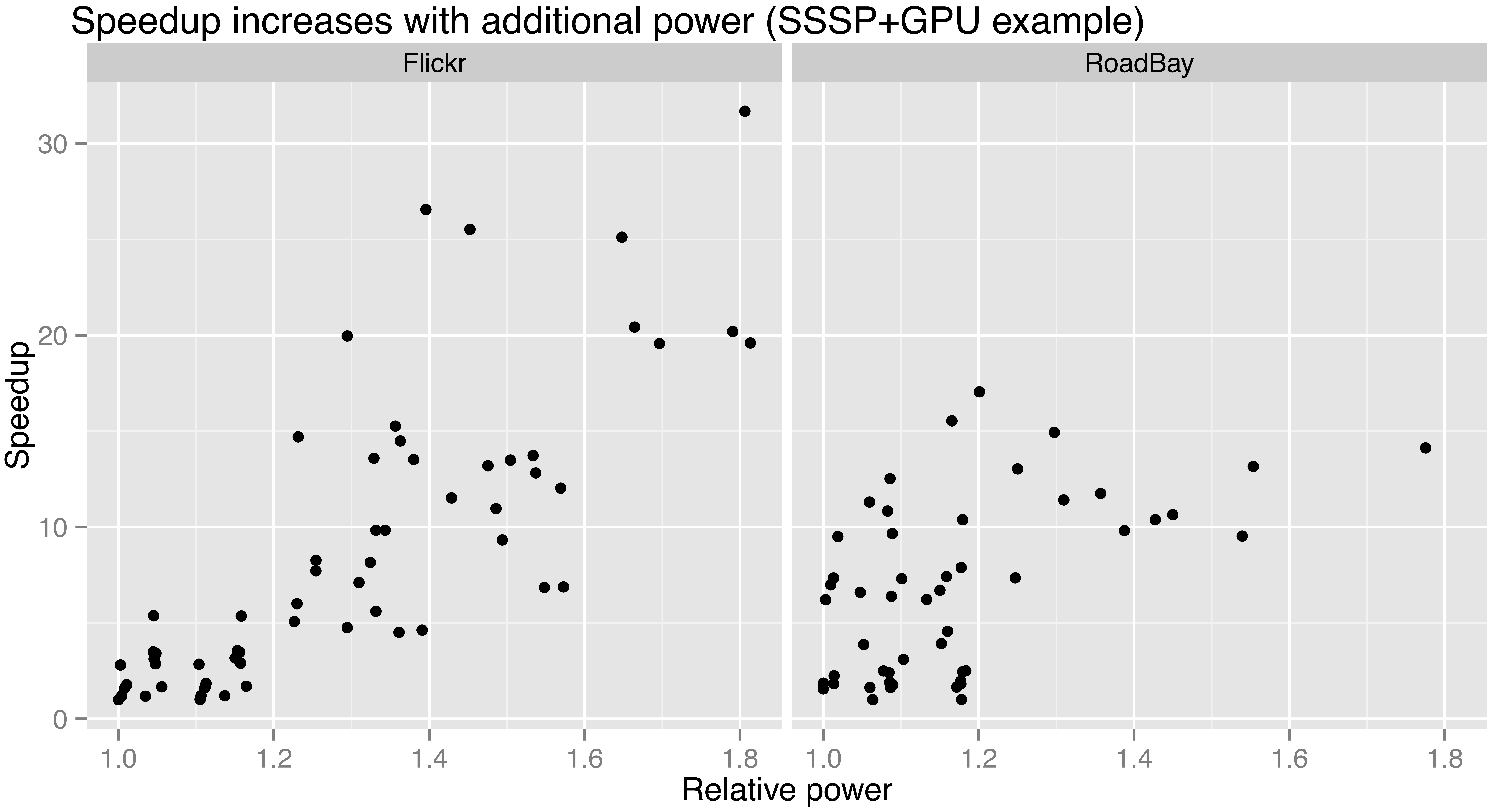


$W(n)$ = *work* (total ops)

$D(n)$ = *span* (critical path)

$W(n)\ /\ D(n)$
= ***inherent parallelism***

⇒ One expects **algorithmic trade-offs** between **time** and **power.**


Speedup increases with additional power (SSSP+GPU example)
Flickr
RoadBay
Speedup
Relative power
0
10
20
30
1.0
1.2
1.4
1.6
1.8

Summary so far:

- Energy optimality ~ work optimality
- Time ~ energy
- Time and power trade-off

Assumes uniform time and energy costs**.**

What if the costs are **non-uniform?**

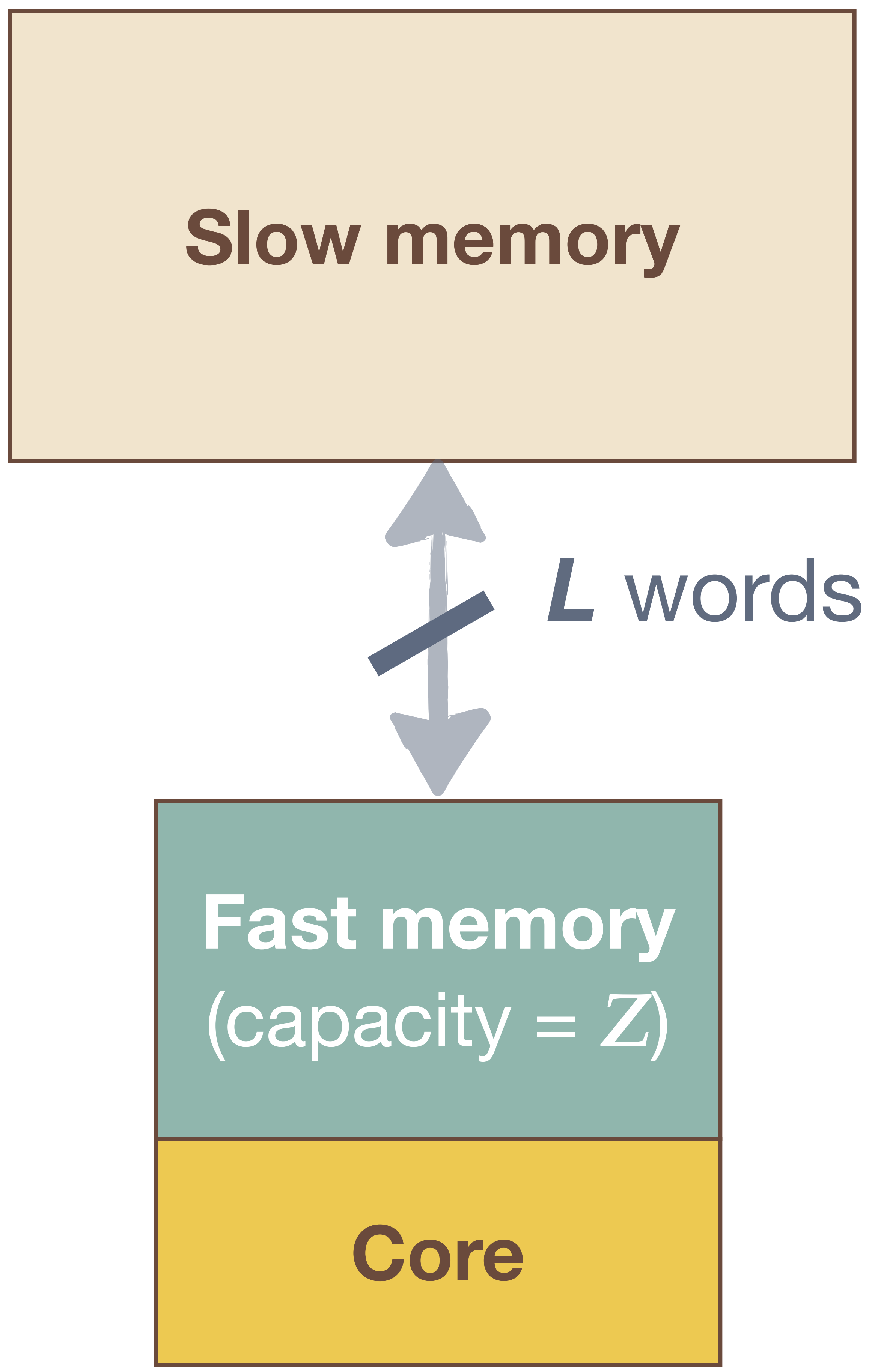
(At least) two interesting cases:

1. **Operation ("op") costs may change**, e.g., under DVFS.
2. **Ops differ in cost**, e.g., computation vs. communication.

(At least) two interesting cases:

1. **Operation (“op”) costs may change**, e.g., under DVFS.
2. **Ops differ in cost**, e.g., computation vs. communication.

Example: **External memory model**



$W(n)$ = *work* (total ops)

$Q(n; Z, L)$ = *no. transfers*

$W(n) \ / \ (Q(n; Z, L) \cdot L)$
= **computational intensity**
(ops / words)

Desiderata:

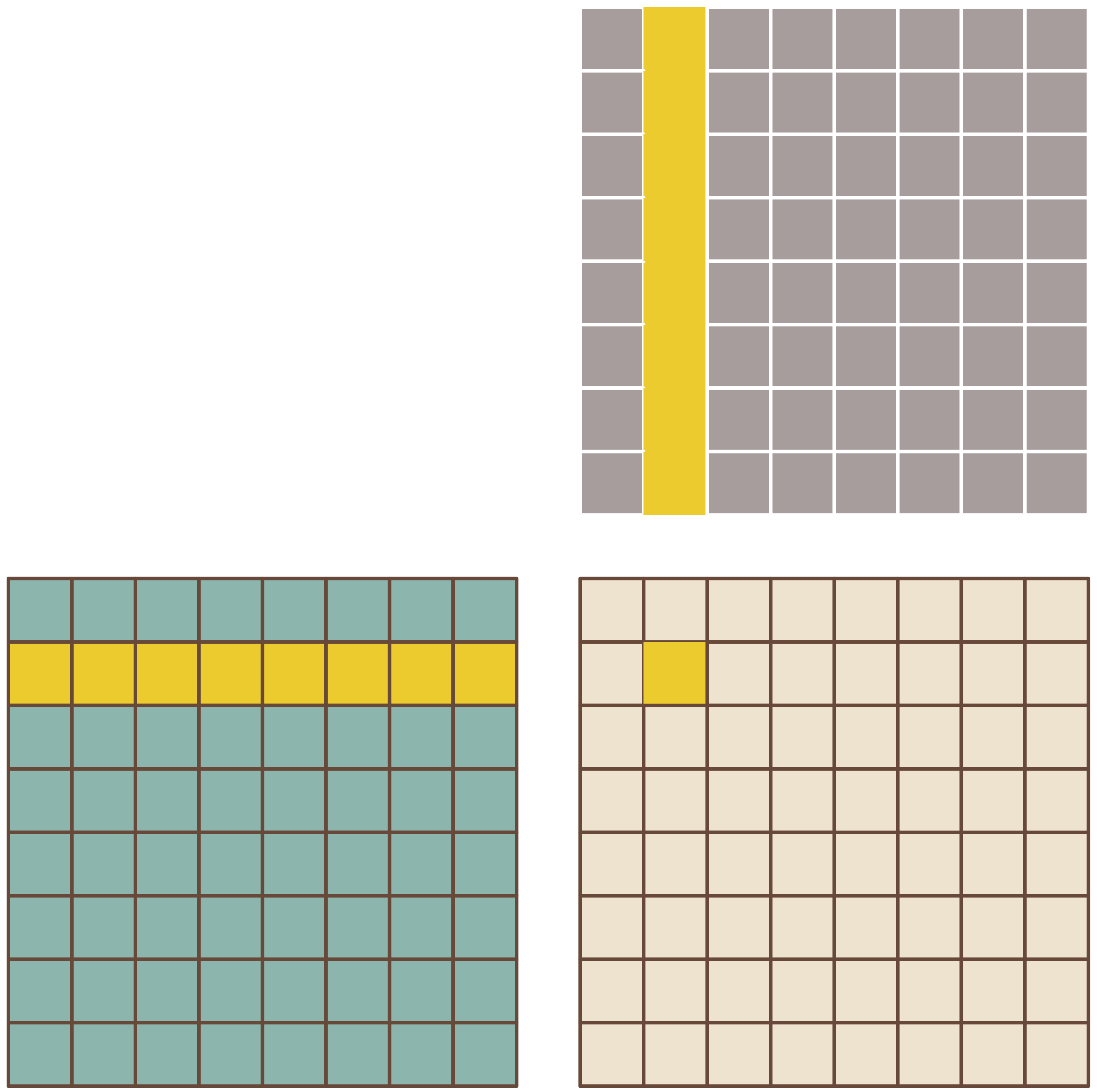
Work optimality

Maximal intensity

E.g., Agarwal and Vitter (1988), …

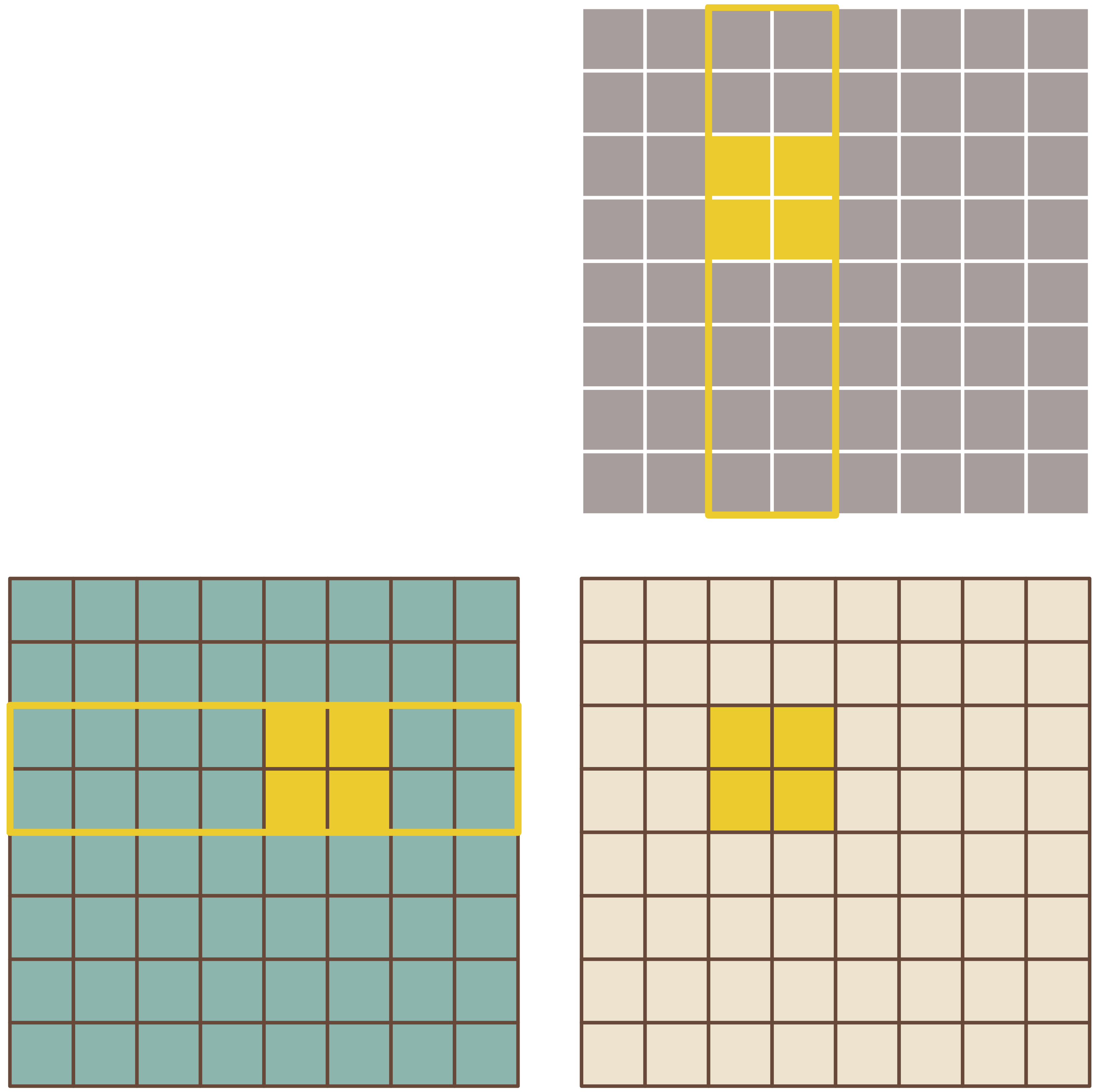
Example: **Matrix multiply**

(non-Strassen)



C ← C + A * B

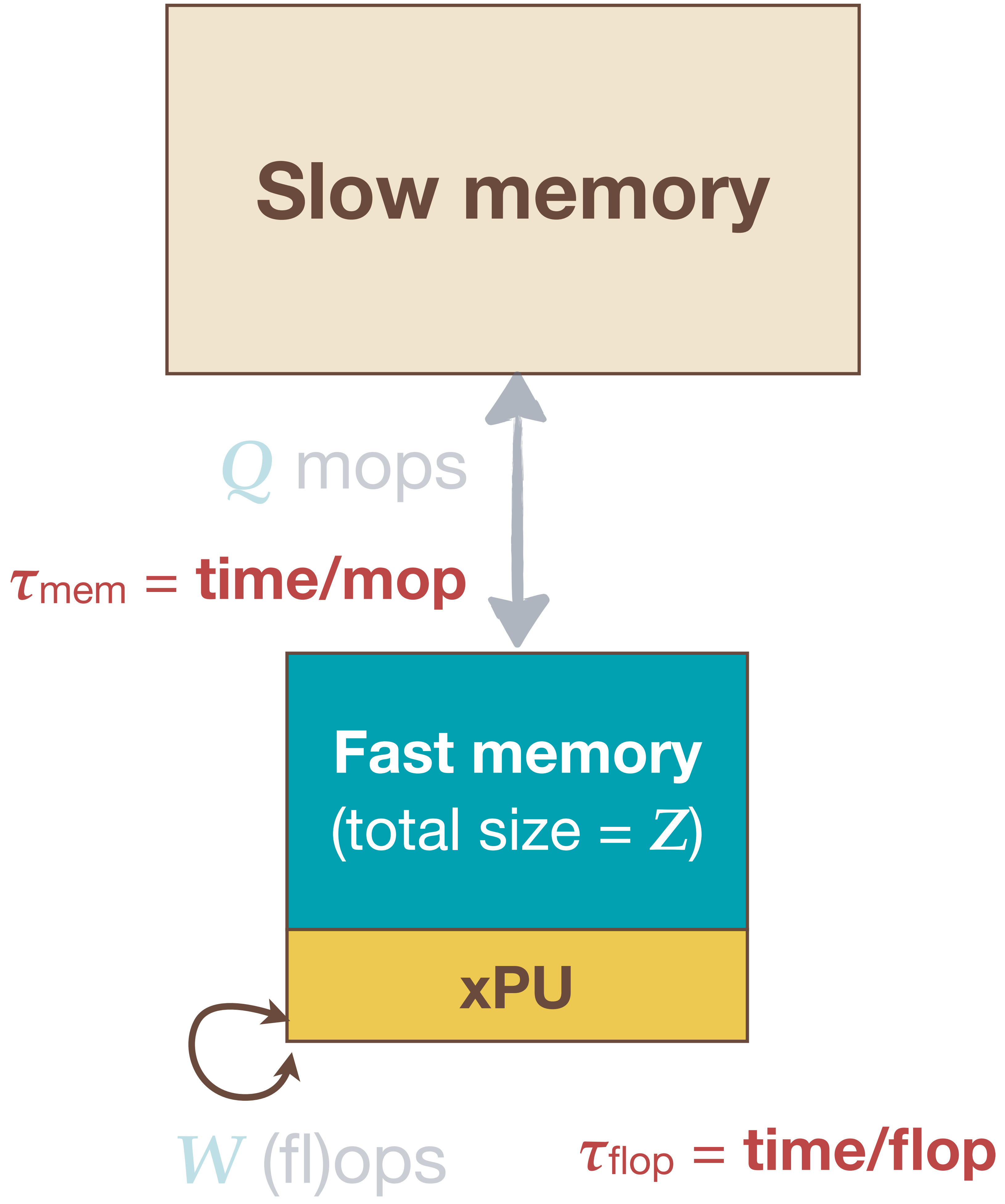
$$\frac{W(n)}{Q(n;Z,L)\cdot L} = \Theta(1)$$



C ← C + A * B

$$\frac{W}{Q\cdot L} = \mathcal{O}\left(\sqrt{Z}\right)$$

Intensity



$$W \equiv \#\text{ (fl)ops}$$

$$Q \equiv \#\text{ mem. ops (mops)} = Q(Z)$$

$$I \equiv \frac{W}{Q} = \text{Intensity (flop:mop)}$$

$$\tau_{\text{flop}} \equiv \text{time per (fl)op}$$

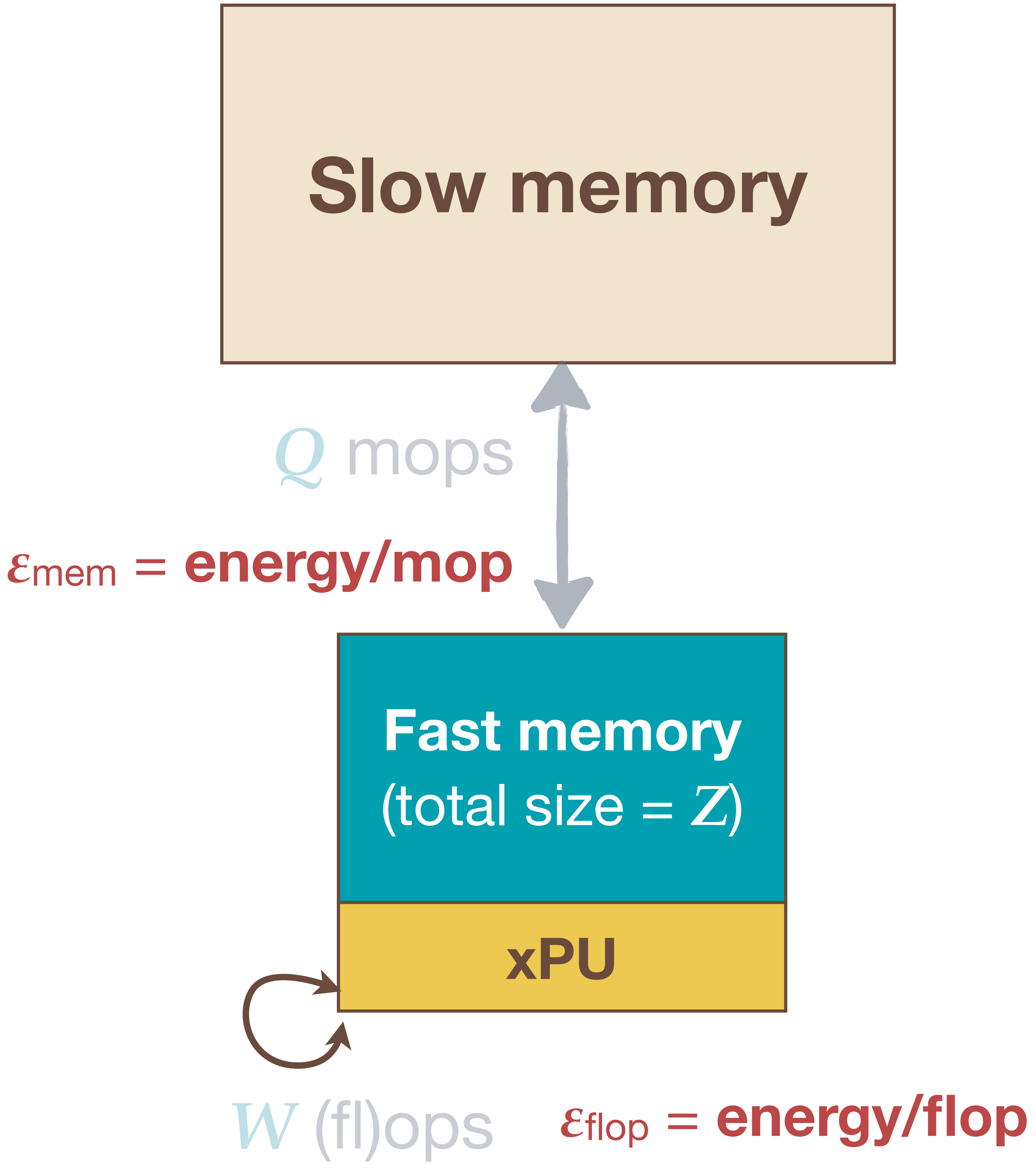
$$\tau_{\text{mem}} \equiv \text{time per mop}$$

$$B_\tau \equiv \frac{\tau_{\text{mem}}}{\tau_{\text{flop}}} = \text{Balance (flop:mop)}$$

Running time

Balance analysis — Kung (1986); Hockney & Curington (1989); Blelloch (1994); McCalpin (1995); Williams et al. (2009); Czechowski et al. (2011); …

hpcgarage.org/ppam15



$$W \equiv \text{\# (fl)ops}$$

$$Q \equiv \text{\# mem. ops (mops)} = Q(Z)$$

$$I \equiv \frac{W}{Q} = \text{Intensity (flop:mop)}$$

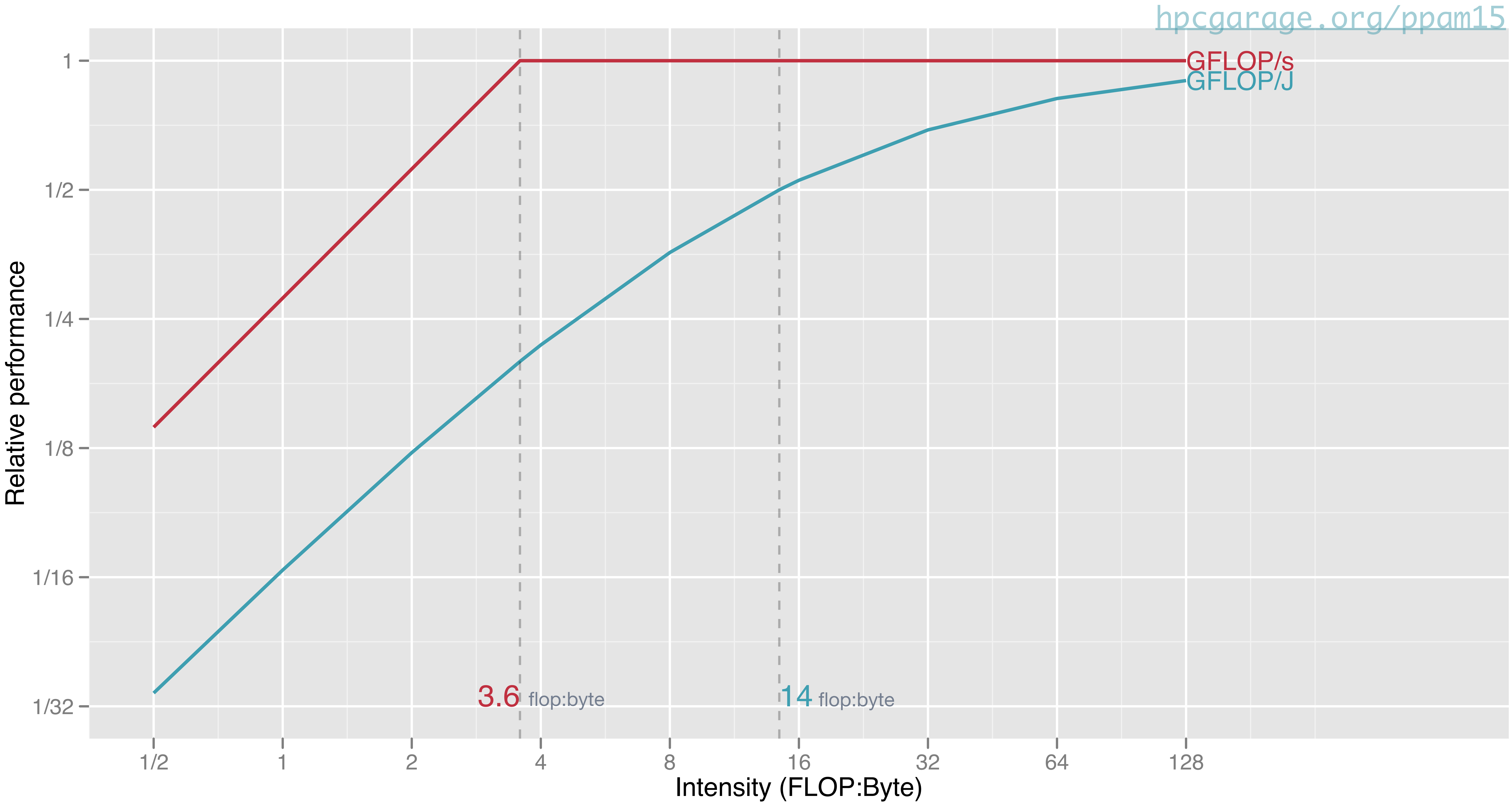
$$\epsilon_{\text{flop}} \equiv \text{energy per (fl)op}$$

$$\epsilon_{\text{mem}} \equiv \text{energy per mop}$$

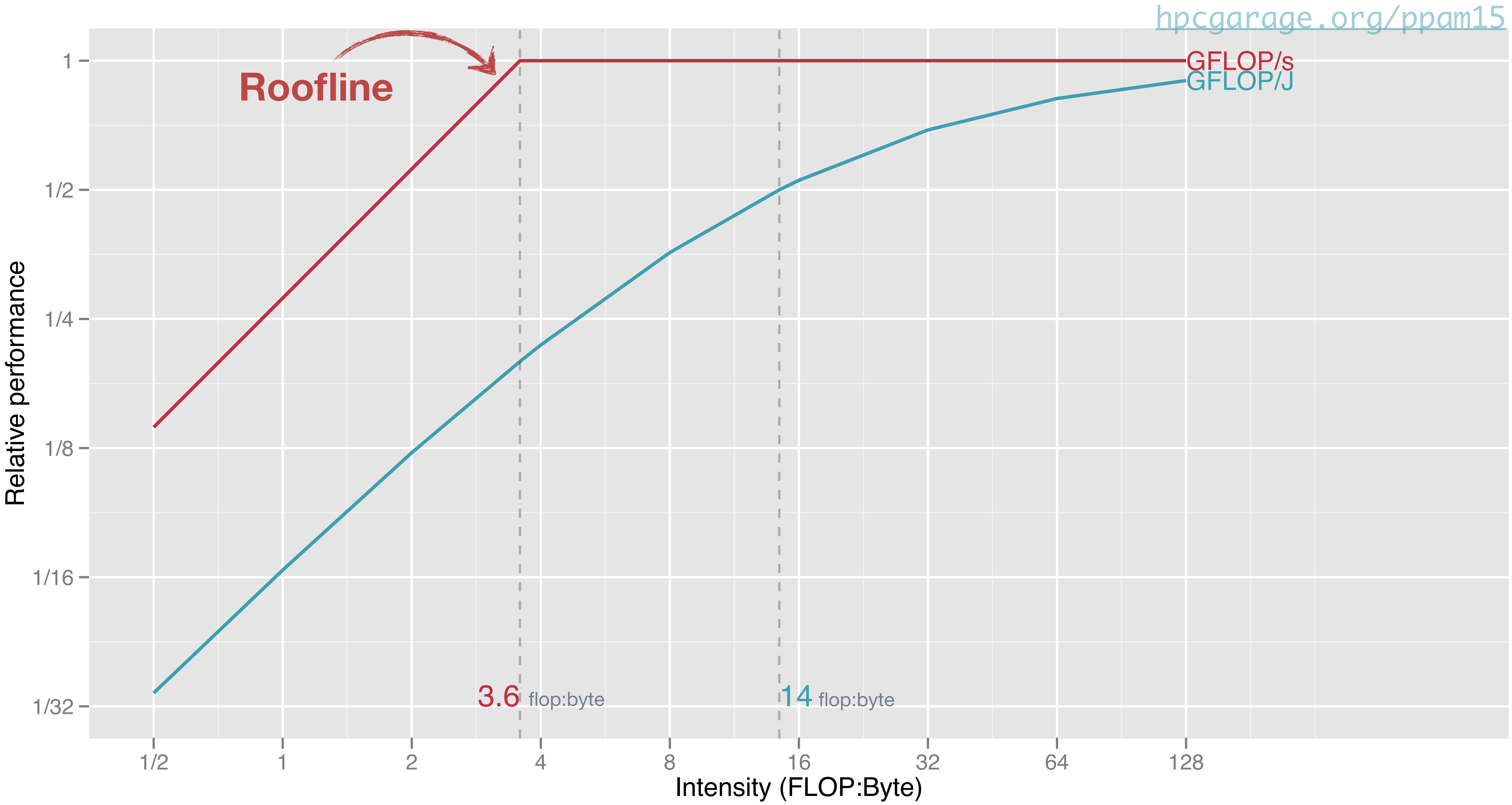
$$B_\epsilon \equiv \frac{\epsilon_{\text{mem}}}{\epsilon_{\text{flop}}} = \text{Energy balance (flop:mop)}$$

Running energy

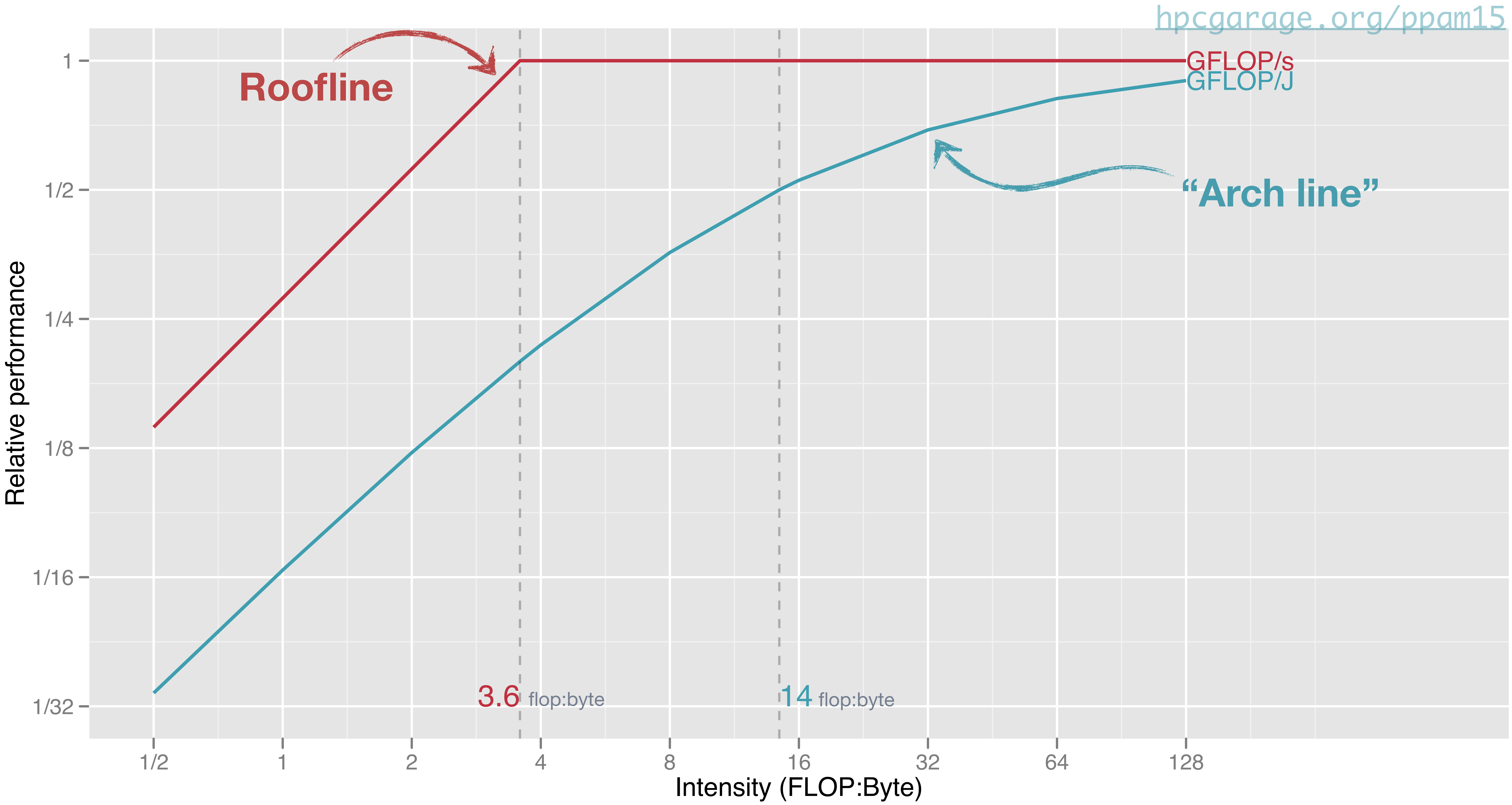
Balance analysis — Kung (1986); Hockney & Curington (1989); Blelloch (1994); McCalpin (1995); Williams et al. (2009); Czechowski et al. (2011); …



J. Choi, D. Bedard, R. Fowler, R. Vuduc. “A roofline model of energy.” *IPDPS* (2013).

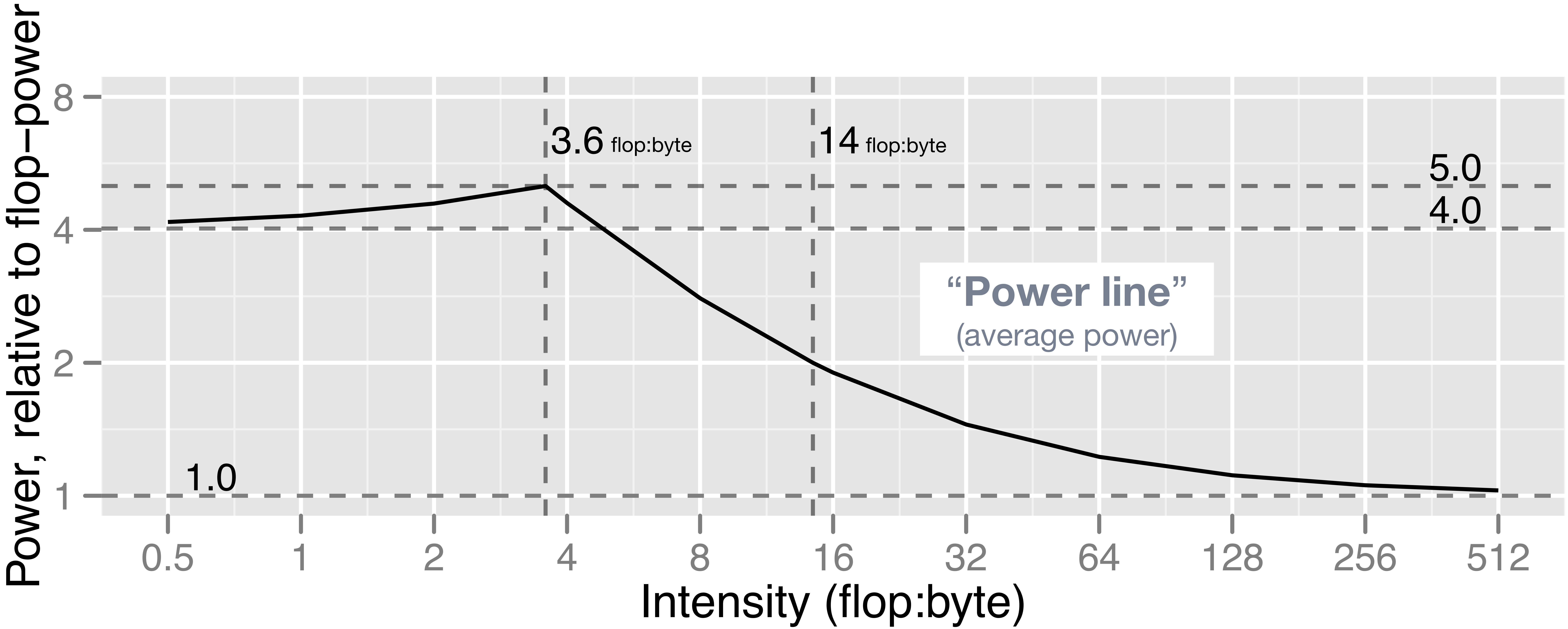


J. Choi, D. Bedard, R. Fowler, R. Vuduc. “A roofline model of energy.” *IPDPS* (2013).



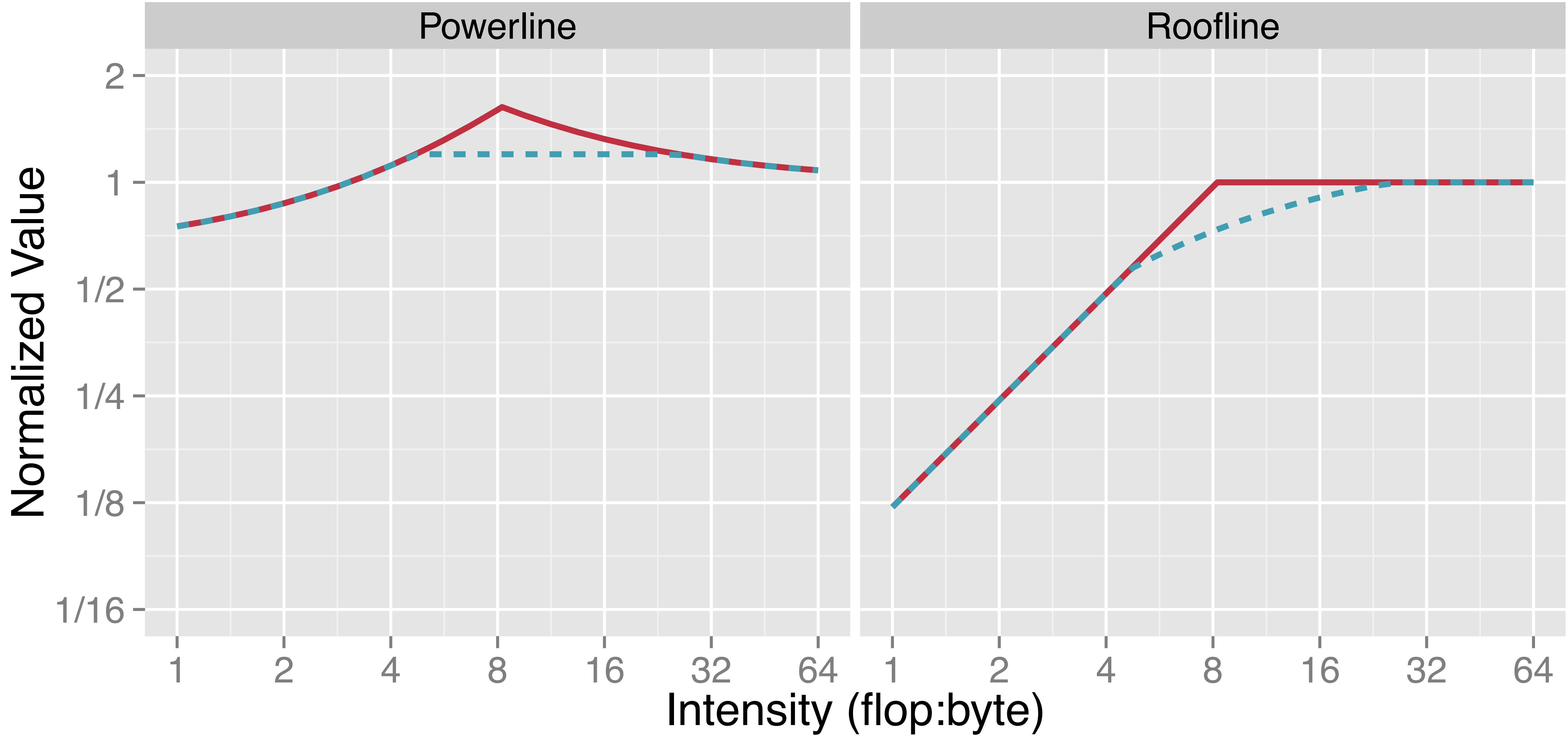
J. Choi, D. Bedard, R. Fowler, R. Vuduc. "A roofline model of energy." *IPDPS* (2013).

hpcgarage.org/ppam15

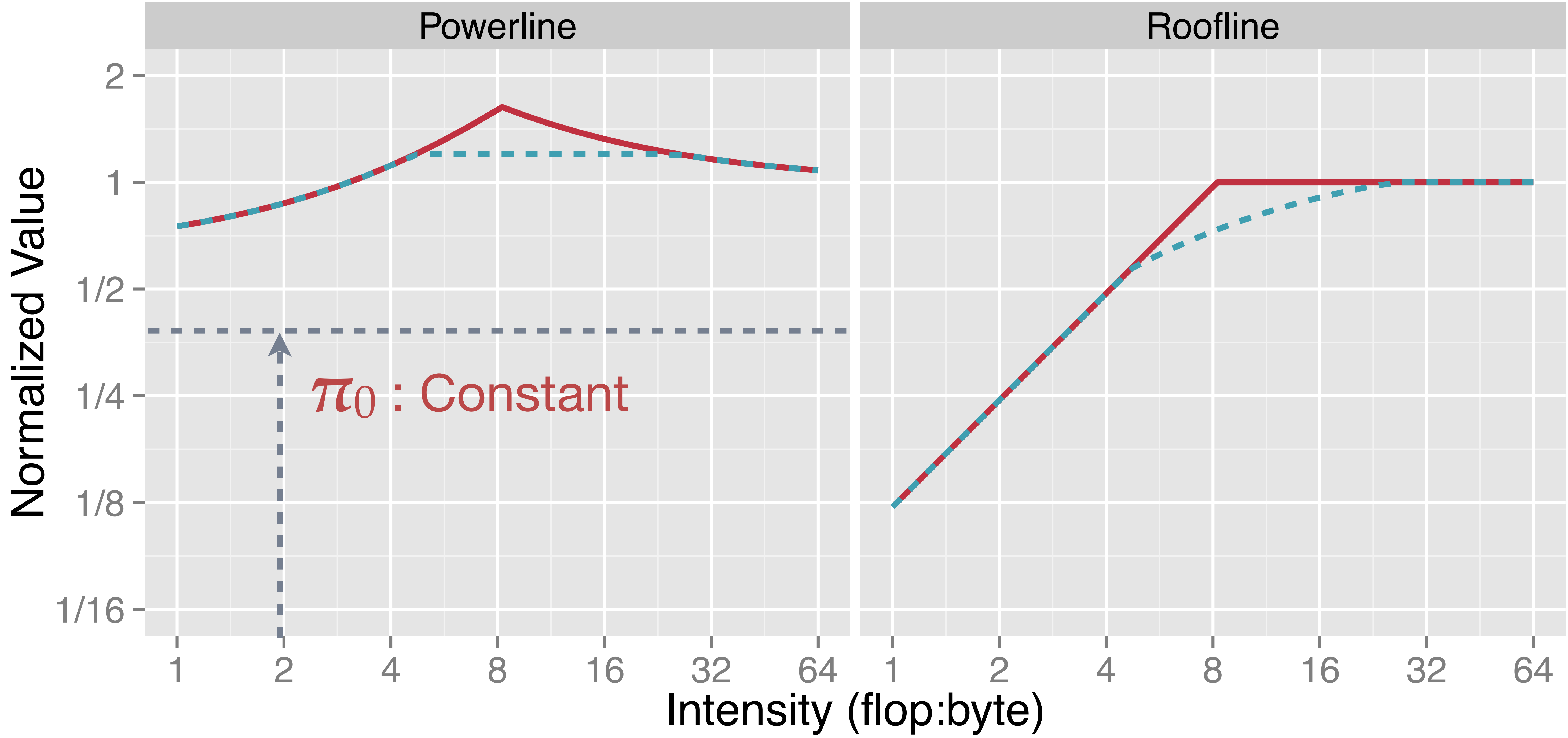


J. Choi, D. Bedard, R. Fowler, R. Vuduc. “A roofline model of energy.” *IPDPS* (2013).

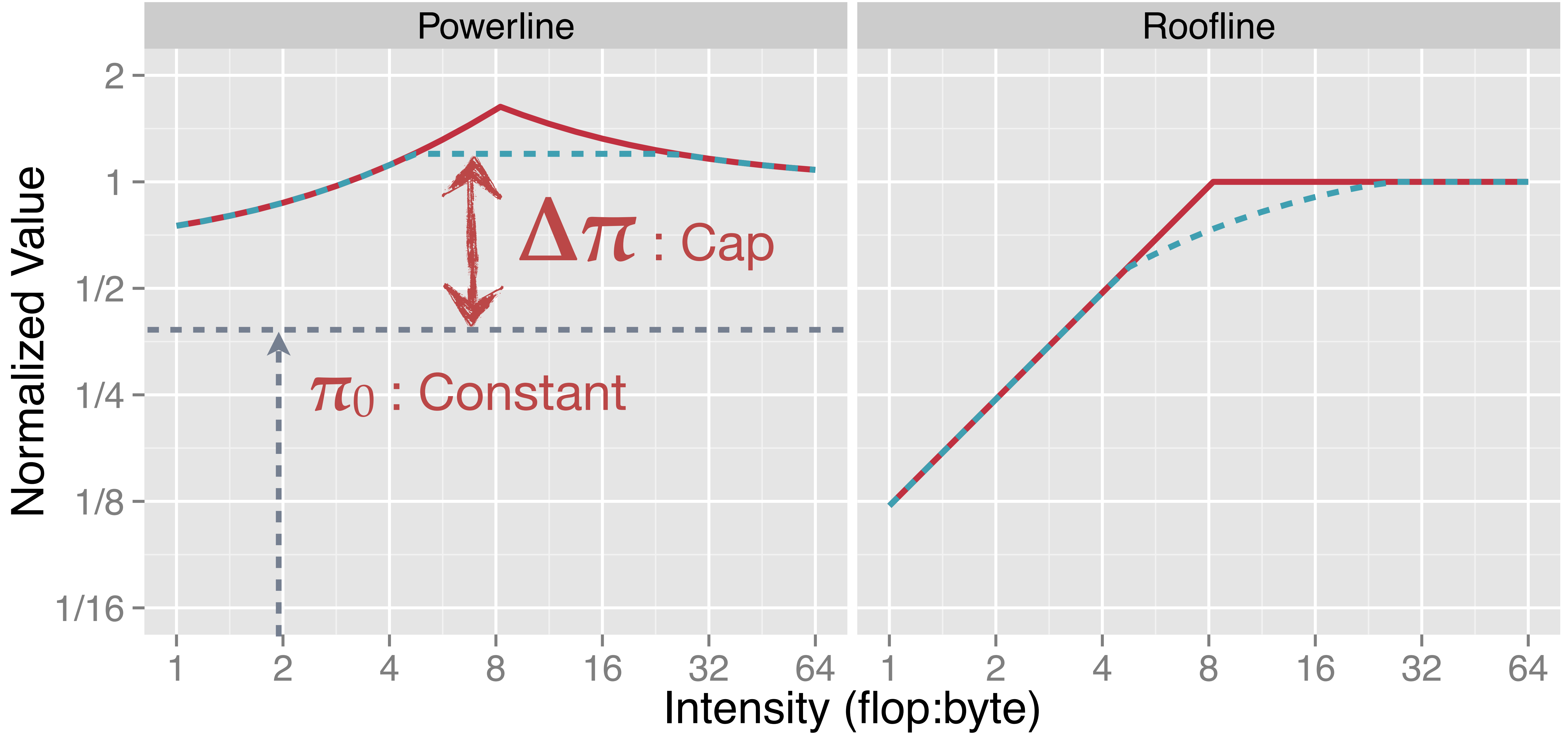
For real systems, the model should account for "**constant power**" and "**power capping**."



For real systems, the model should account for "**constant power**" and "**power capping**."



For real systems, the model should account for "**constant power**" and "**power capping**."




GTX Titan
"Desktop GPU"
16 Gflop/J
4.0 Tflop/s [81%], 240 GB/s [83%]
120 W (const) + 160 W (cap) [99%]
GTX 680
"Desktop GPU"
15 Gflop/J
3.0 Tflop/s [86%], 160 GB/s [82%]
66 W (const) + 140 W (cap) [100%]
Xeon Phi
"Desktop xPU"
11 Gflop/J
2.0 Tflop/s [100%], 180 GB/s [57%]
180 W (const) + 36 W (cap) [100%]
NUC GPU
"Mobile GPU"
8.8 Gflop/J
270 Gflop/s [100%], 15 GB/s [60%]
10 W (const) + 18 W (cap) [91%]
Arndale GPU
"Mobile GPU"
8.1 Gflop/J
33 Gflop/s [46%], 8.4 GB/s [66%]
1.3 W (const) + 4.8 W (cap) [88%]
APU GPU
"APU GPU"
6.4 Gflop/J
100 Gflop/s [95%], 8.7 GB/s [81%]
16 W (const) + 3.2 W (cap) [100%]
GTX 580
"Desktop GPU"
5.4 Gflop/J
1.4 Tflop/s [88%], 170 GB/s [89%]
120 W (const) + 150 W (cap) [94%]
NUC CPU
"Mobile CPU"
3.2 Gflop/J
56 Gflop/s [97%], 18 GB/s [70%]
17 W (const) + 7.4 W (cap) [98%]
PandaBoard ES
"Mobile CPU"
2.5 Gflop/J
9.5 Gflop/s [99%], 1.3 GB/s [40%]
3.5 W (const) + 1.2 W (cap) [95%]
Arndale CPU
"Mobile CPU"
2.2 Gflop/J
16 Gflop/s [58%], 3.9 GB/s [31%]
5.5 W (const) + 2.0 W (cap) [97%]
APU CPU
"APU CPU"
650 Mflop/J
13 Gflop/s [98%], 3.3 GB/s [31%]
20 W (const) + 1.4 W (cap) [98%]
Desktop CPU
"Desktop CPU"
620 Mflop/J
99 Gflop/s [93%], 19 GB/s [74%]
120 W (const) + 44 W (cap) [99%]
Memory
Cap
Compute
Power [normalized]
Intensity (single−precision flop:Byte)
1/8
1/2
2
8
32
128
512
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
1
1.05
1.1
1.15

“Desktop GPU”
“Mobile GPU”
GTX Titan
Arndale GPU
Power [normalized]
Intensity (single−precision flop:Byte)
Cap
Memory
Compute
16 Gflop/J, 1.3 GB/J
4.0 Tflop/s [81%], 240 GB/s [83%]
120 W (const) + 160 W (cap) [99%]
8.1 Gflop/J, 1.5 GB/J
33 Gflop/s [46%], 8.4 GB/s [66%]
1.3 W (const) + 4.8 W (cap) [88%]
1.15
1.1
1.05
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
1/8
1/2
2
8
32
128
512

GTX Titan
GTX 680
Xeon Phi
NUC GPU
Arndale GPU
APU GPU
“Desktop GPU”
“Desktop GPU”
“Desktop xPU”
“Mobile GPU”
“Mobile GPU”
“APU GPU”
GTX Titan
Arndale GPU
Power [normalized]
1.15
1.1
1.05
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
Cap
Compute
Memory
16 Gflop/J, 1.3 GB/J
4.0 Tflop/s [81%], 240 GB/s [83%]
120 W (const) + 160 W (cap) [99%]
Memory
Cap
Compute
8.1 Gflop/J, 1.5 GB/J
33 Gflop/s [46%], 8.4 GB/s [66%]
1.3 W (const) + 4.8 W (cap) [88%]
1/8
1/2
2
8
32
128
512
Intensity (single−precision flop:Byte)
1.4 Tflop/s [88%], 170 GB/s [89%]
120 W (const) + 150 W (cap) [94%]
56 Gflop/s [97%], 18 GB/s [70%]
17 W (const) + 7.4 W (cap) [98%]
9.5 Gflop/s [99%], 1.3 GB/s [40%]
3.5 W (const) + 1.2 W (cap) [95%]
16 Gflop/s [58%], 3.9 GB/s [31%]
5.5 W (const) + 2.0 W (cap) [97%]
13 Gflop/s [98%], 3.3 GB/s [31%]
20 W (const) + 1.4 W (cap) [98%]
620 Mflop/J
99 Gflop/s [93%], 19 GB/s [74%]
120 W (const) + 44 W (cap) [99%]
6.4 Gflop/J

Work-communication trade-offs

Abstract work-communication trade-offs

Algorithm 1 $= (W, Q)$ versus Algorithm 2 $= (fW, \frac{Q}{m})$

$$I \equiv \frac{W}{Q}$$

Abstract work-communication trade-offs

$$\text{Algorithm 1} = (W, Q) \quad \text{versus} \quad \text{Algorithm 2} = (fW, \frac{Q}{m})$$

$$I \equiv \frac{W}{Q}$$

More work

Less communication

Abstract work-communication trade-offs

Algorithm 1 $= (W, Q)$ versus Algorithm 2 $= (fW, \frac{Q}{m})$

$$I \equiv \frac{W}{Q}$$

More work

Less communication

Speedup $\Delta T = \frac{T_{1,1}}{T_{f,m}}$

"Greenup" $\Delta E = \frac{E_{1,1}}{E_{f,m}}$

Abstract work-communication trade-offs

Algorithm 1 $= (W, Q)$ versus Algorithm 2 $= (fW, \frac{Q}{m})$

More work

Less communication

$$I \equiv \frac{W}{Q}$$

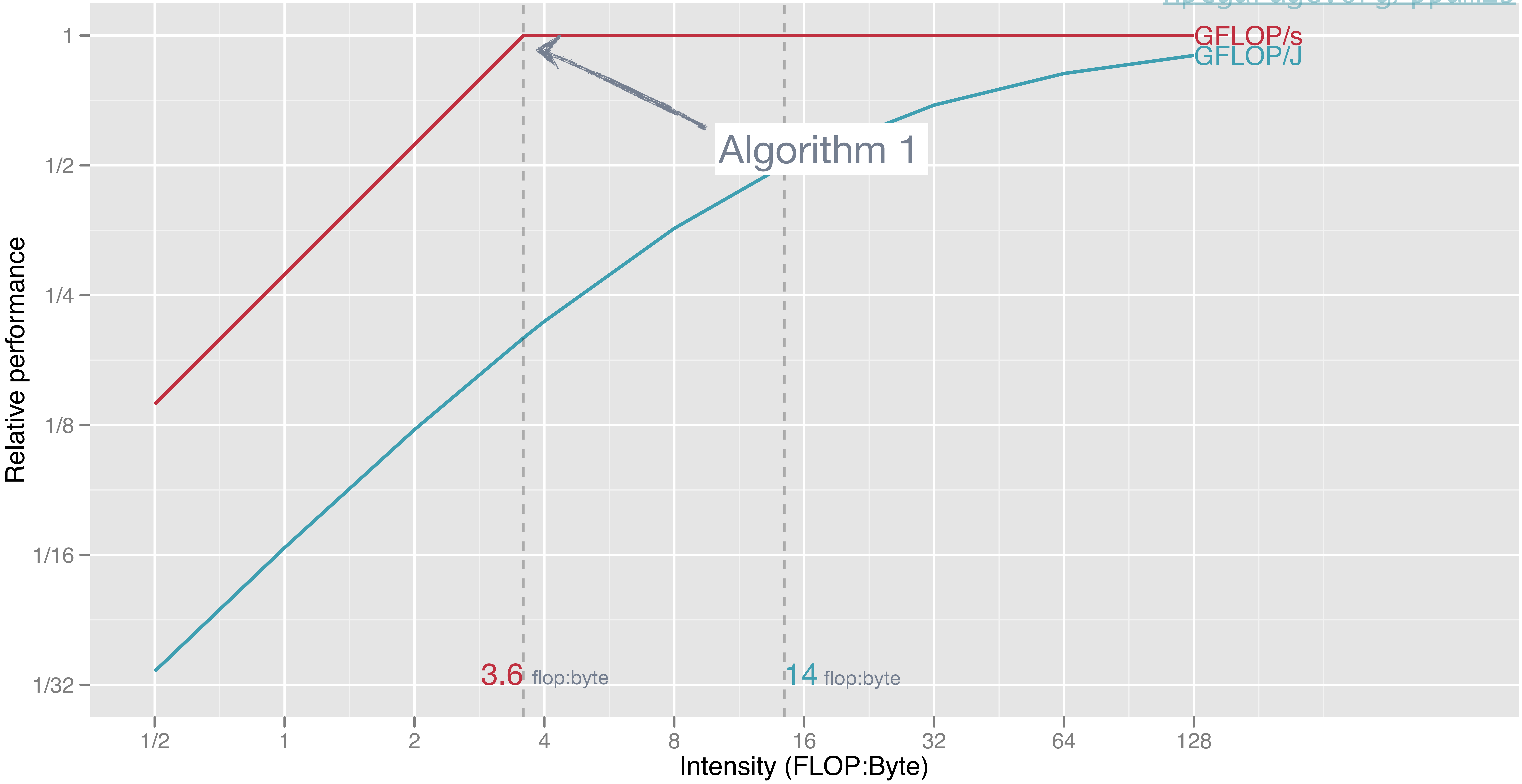
$$\text{Speedup } \Delta T = \frac{T_{1,1}}{T_{f,m}}$$

$$\text{“Greenup” } \Delta E = \frac{E_{1,1}}{E_{f,m}}$$

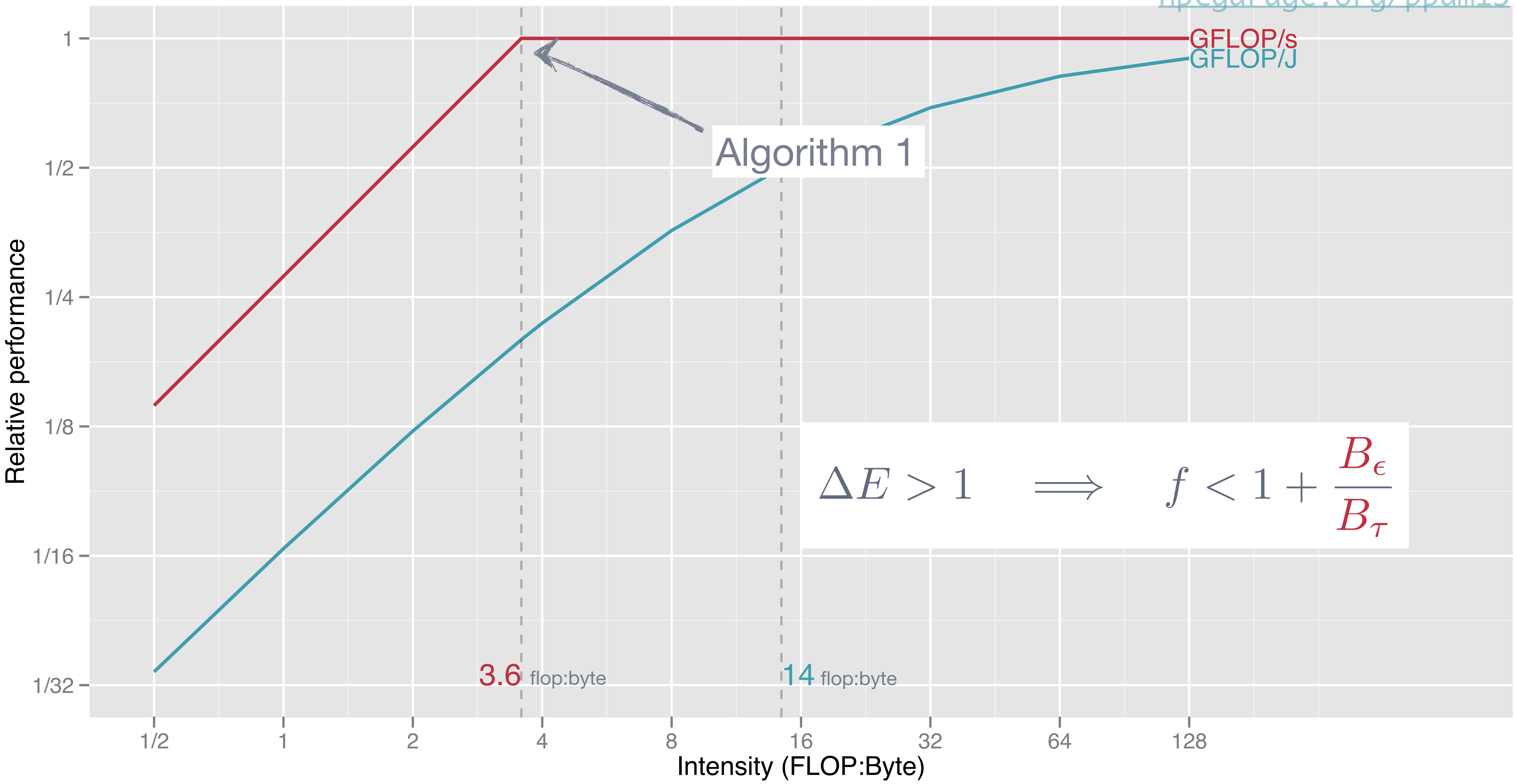
$$\boxed{\Delta E > 1 \implies f < 1 + \frac{m-1}{m}\frac{B_\epsilon}{I}}$$

A general “greenup” condition

hpcgarage.org/ppam15



Balance estimates for a high-end NVIDIA Fermi in *double-precision*, according to Keckler et al. *IEEE Micro* (2011)



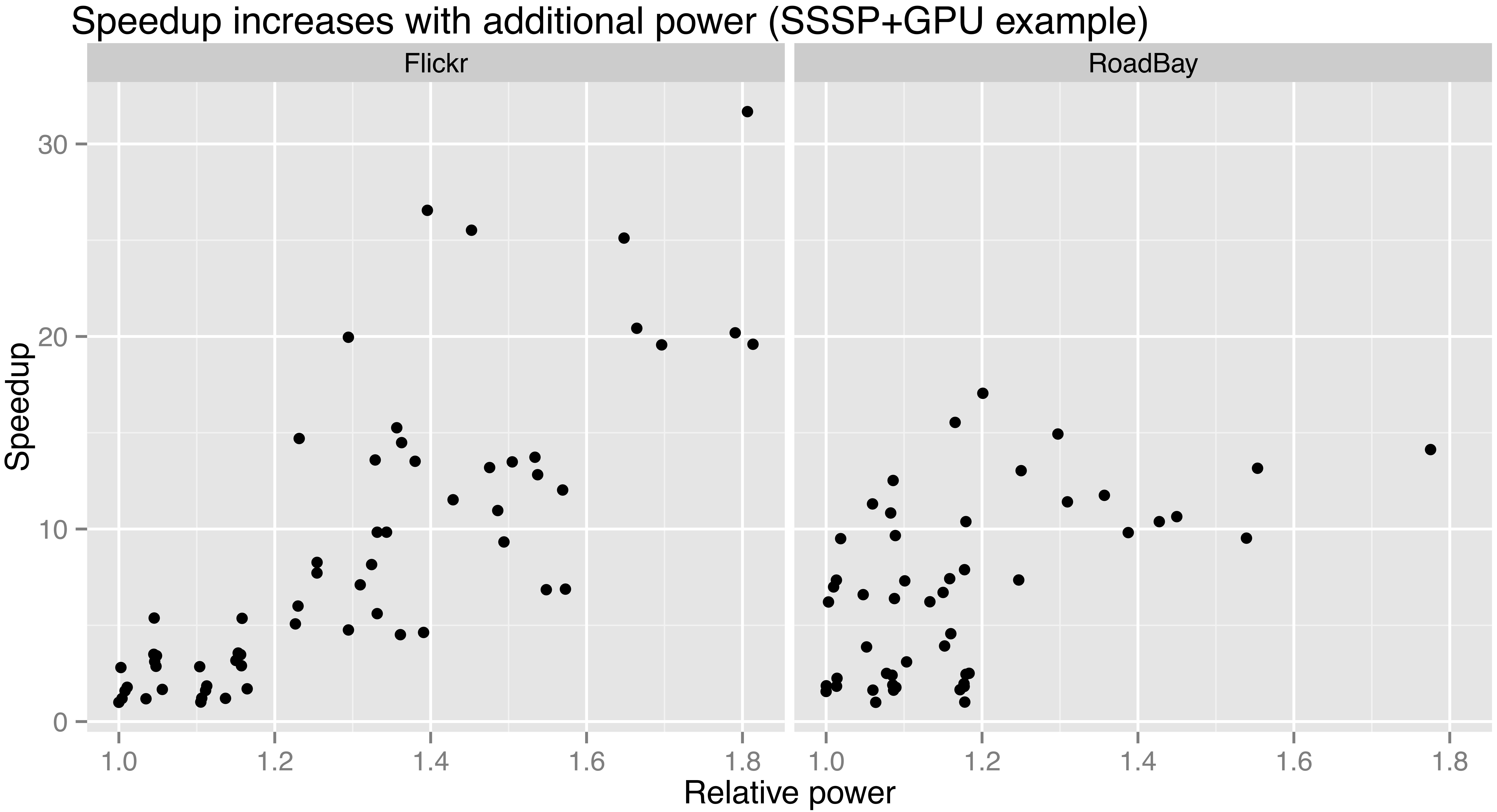
Balance estimates for a high-end NVIDIA Fermi in *double-precision*, according to Keckler et al. *IEEE Micro* (2011)

III. Case study: **Single-source shortest path**

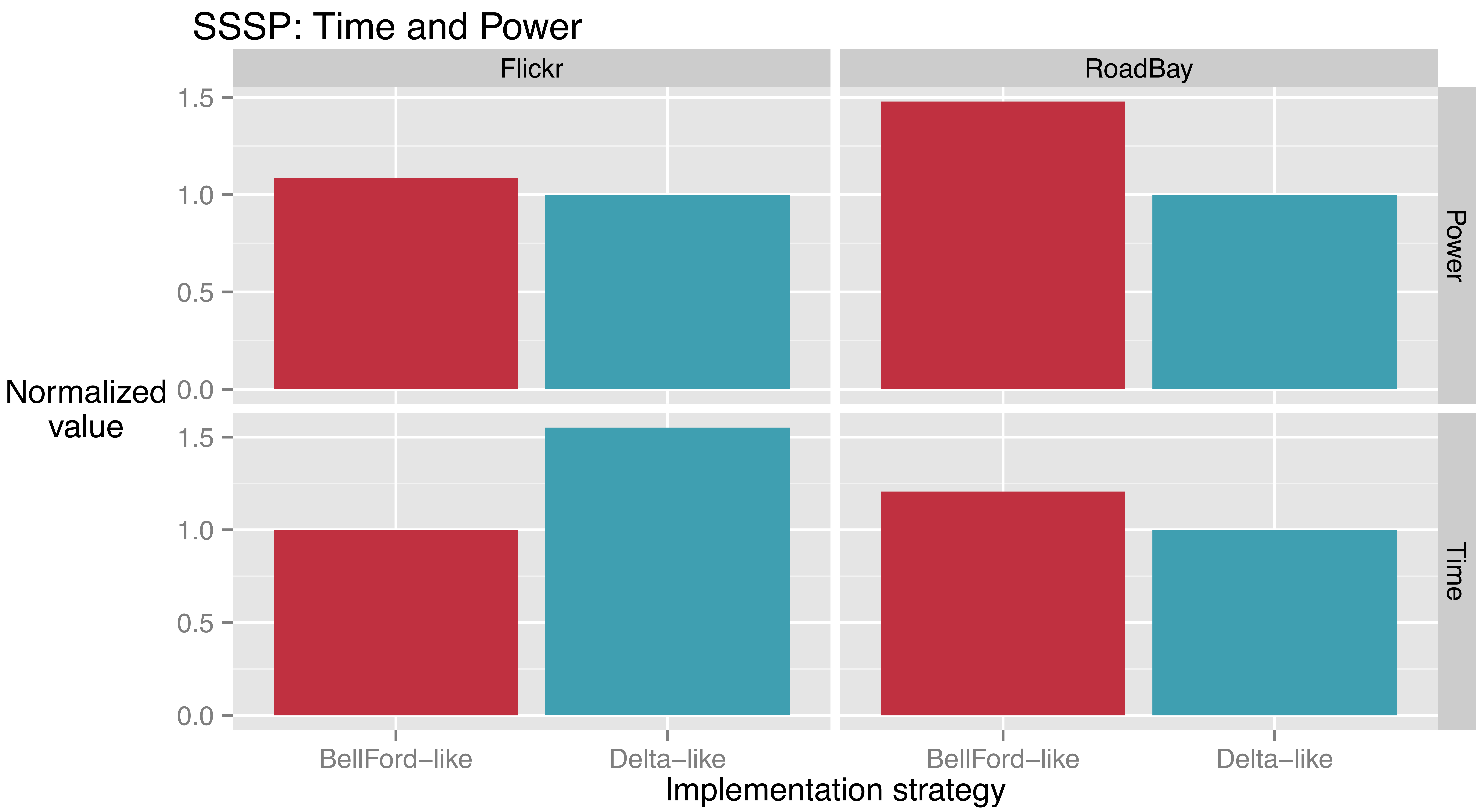
Sara Karamati, Jeff Young (PhD), R. Vuduc

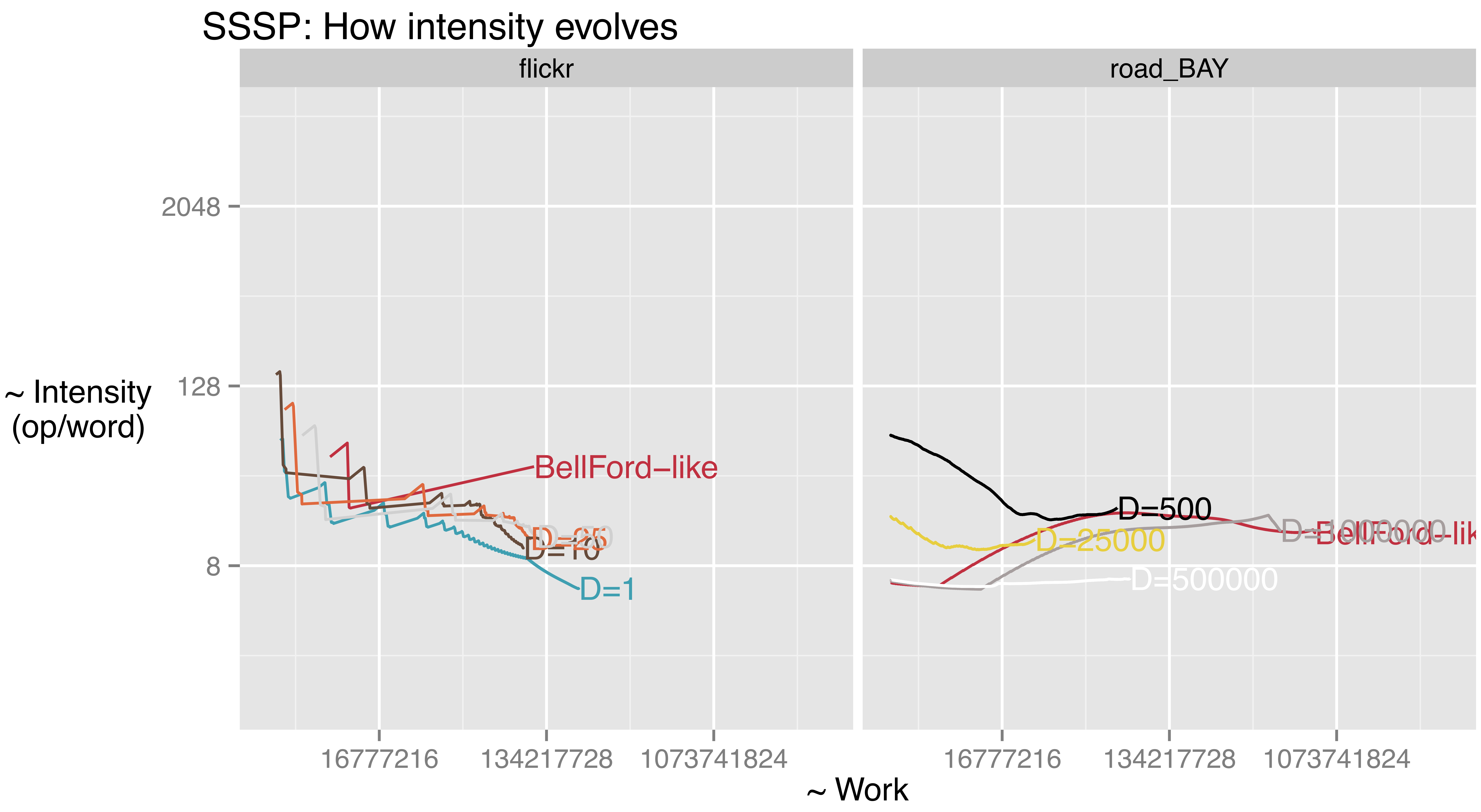
- Consider two implementation variants*
  - "Bellman-Ford-like" — Highly parallel but not work-optimal
  - "Delta-stepping-like" — Tunable work-parallelism tradeoff
  - *No* preprocessing shortcuts, a la PHAST**
- Both are tuned* for a GPU and we run them on an NVIDIA Jetson TK1, which has tunable core frequencies (**10x**) and memory frequencies (**3x**)

** These are GunRock implementations of Davidson, Baxter, Garland, and Owens (IPDPS'14)*
*** Delling et al. "PHAST: Hardware-accelerated shortest path trees" (JPDC'10)*



Speedup increases with additional power (SSSP+GPU example)
Flickr
RoadBay
Speedup
0
10
20
30
1.0
1.2
1.4
1.6
1.8
Relative power

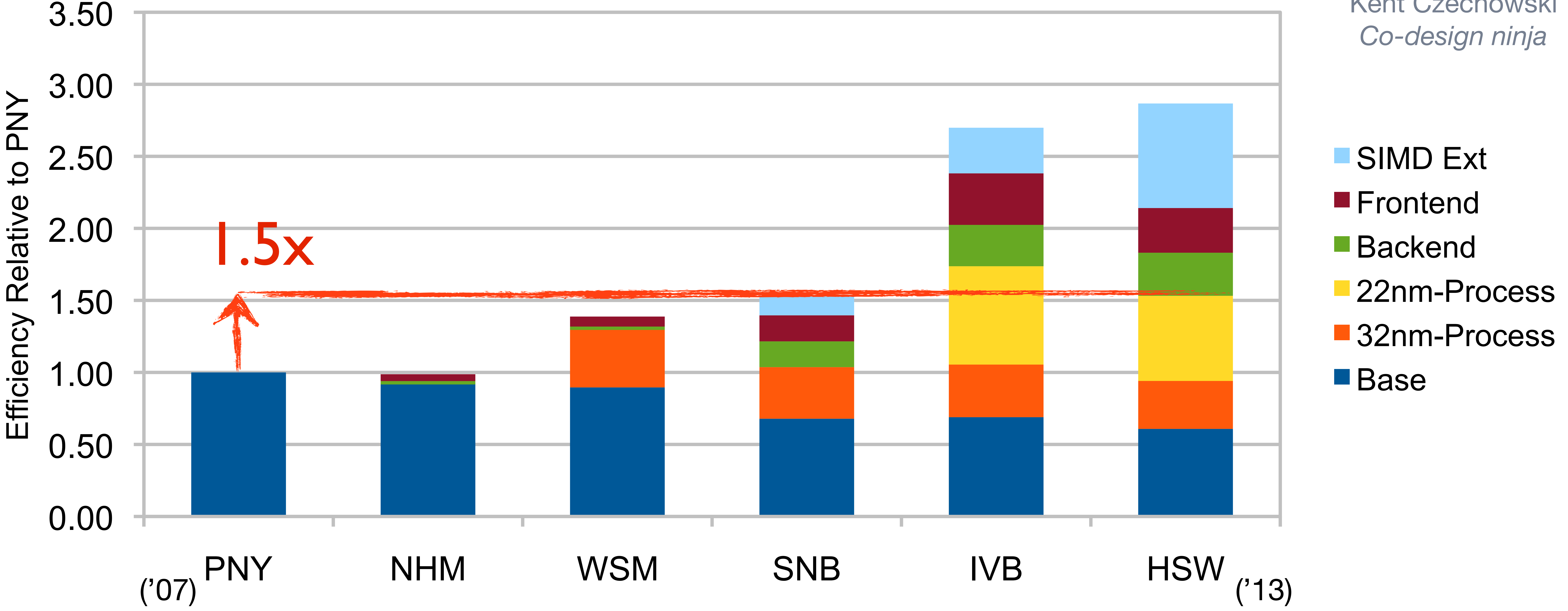

SSSP: Time and Power
Flickr
RoadBay
Power
Time
Normalized value
1.5
1.0
0.5
0.0
BellFord–like
Delta–like
Implementation strategy


SSSP: How intensity evolves
flickr
road_BAY
~ Intensity (op/word)
2048
128
8
16777216
134217728
1073741824
~ Work
BellFord–like
D=10
D=1
D=500
D=25000
D=500000

IV. Case study: **Branch-avoidance**

Oded Green, Marat Dukhan, RV. "Branch-avoiding graph algorithms." In SPAA'15.
+ Post-paper analysis help from Anita Zakrzewska

Improvement in Energy Efficiency
Livermore Loops



Kent Czechowski
Co-design ninja

K. Czechowski et al. "Improving the energy-efficiency of big cores." In ISCA'14.

Shiloach-Vishkin algorithm to compute connected components (as labels)

```
forall v ∈ V do
  label[v] ← int(v)

while … do
  forall v ∈ V do
    forall (v, u) ∈ E do
      if label[v] < label[u] then
        label[v] ← label[u]
```

O. Green, M. Dukhan, R. Vuduc. "Branch-avoiding graph algorithms." In SPAA'15.

Predicted Cycles per instruction [Ivy Bridge]
astro−ph
audikw1
auto
coAuthorsDBLP
cond−mat−2003
cond−mat−2005
coPapersDBLP
ecology1
ldoor
power
eferentialAttachme
sv
1.2
0.9
0.6
0.3
0.0
Branch−based
Predicted values:
Cache.references
Cache.misses
Branches
Mispredictions

Predicted Cycles per instruction [Ivy Bridge]
astro−ph
audikw1
auto
coAuthorsDBLP
cond−mat−2003
cond−mat−2005
coPapersDBLP
ecology1
ldoor
power
eferentialAttachme
SV
1.2
0.9
0.6
0.3
0.0
Measured
Branch−based
Predicted values:
Cache.references
Cache.misses
Branches
Mispredictions


Predicted Cycles per instruction [Ivy Bridge]
astro−ph
audikw1
auto
coAuthorsDBLP
cond−mat−2003
cond−mat−2005
coPapersDBLP
ecology1
ldoor
power
eferentialAttachme
sv
1.2
0.9
0.6
0.3
0.0
Branch−based
Modeled
(counters + lasso regression)
Predicted values:
Cache.references
Cache.misses
Branches
Mispredictions

Predicted Cycles per instruction [Ivy Bridge]
astro−ph
audikw1
auto
coAuthorsDBLP
cond−mat−2003
cond−mat−2005
coPapersDBLP
ecology1
ldoor
power
eferentialAttachme
SV
1.2
0.9
0.6
0.3
0.0
Branch−based
Predicted values:
Cache.references
Cache.misses
Branches
Mispredictions

Branch-based (original):

```
forall v ∈ V do
  label[v] ← int(v)

while … do
  forall v ∈ V do
    forall (v, u) ∈ E do
      if label[u] < label[v] then
        label[v] ← label[u]
```

Branch-avoiding:

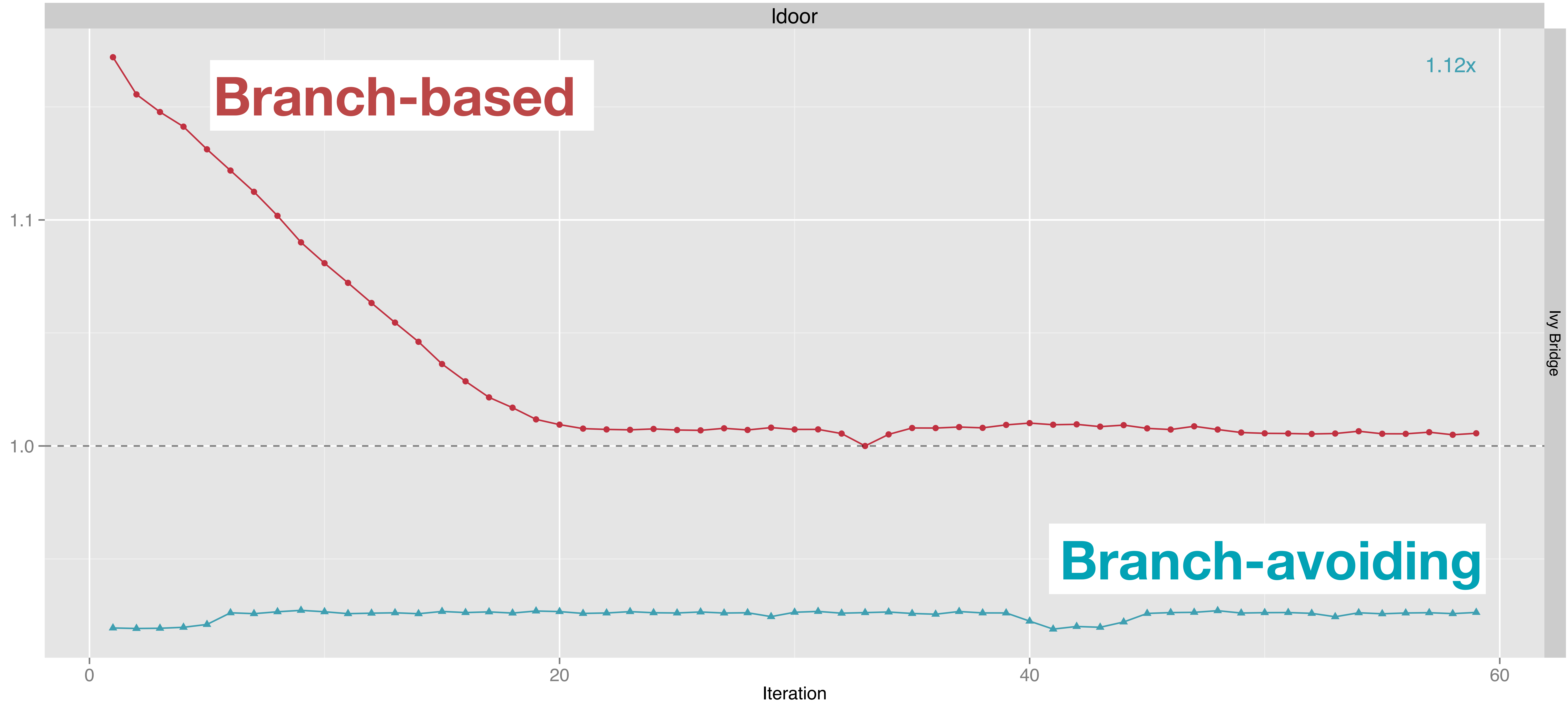
```
forall v ∈ V do
  label[v] ← int(v)

while … do
  forall v ∈ V do
    forall (v, u) ∈ E do
      flag ← (label[u] < label[v])
      cmov (label[v], label[u], flag)
```

O. Green, M. Dukhan, R. Vuduc. "Branch-avoiding graph algorithms." In SPAA'15.

Shiloach–Vishkin Connected Components: Cycles

[Normalized to branch–based minimum]



O. Green, M. Dukhan, R. Vuduc. "Branch-avoiding graph algorithms." In SPAA'15.

Summary

- The key high-level claim of this talk is that **"classical" principles** of algorithm and software design are not only relevant, but **even more important** when considering metrics beyond time, such as energy and power.
- My main suggested direction for **future research** is to explore **work-X** tradeoffs, where X in this talk included communication, parallelism, and branching behavior, but there may be many others!