hpcgarage.org/isc15

Actively analyzing performance

to find microarchitectural bottlenecks and to estimate performance bounds

Kenneth (Kent) Czechowski · Jee Whan Choi (IBM) · Jeff Young · Richard (Rich) Vuduc

July 16, 2015
Workshop on Performance Modeling: Methods and Applications
at International Supercomputing Conference (ISC)

Georgia Tech College of Computing
Computational Science and Engineering

hpcgarage

Kent Czechowski

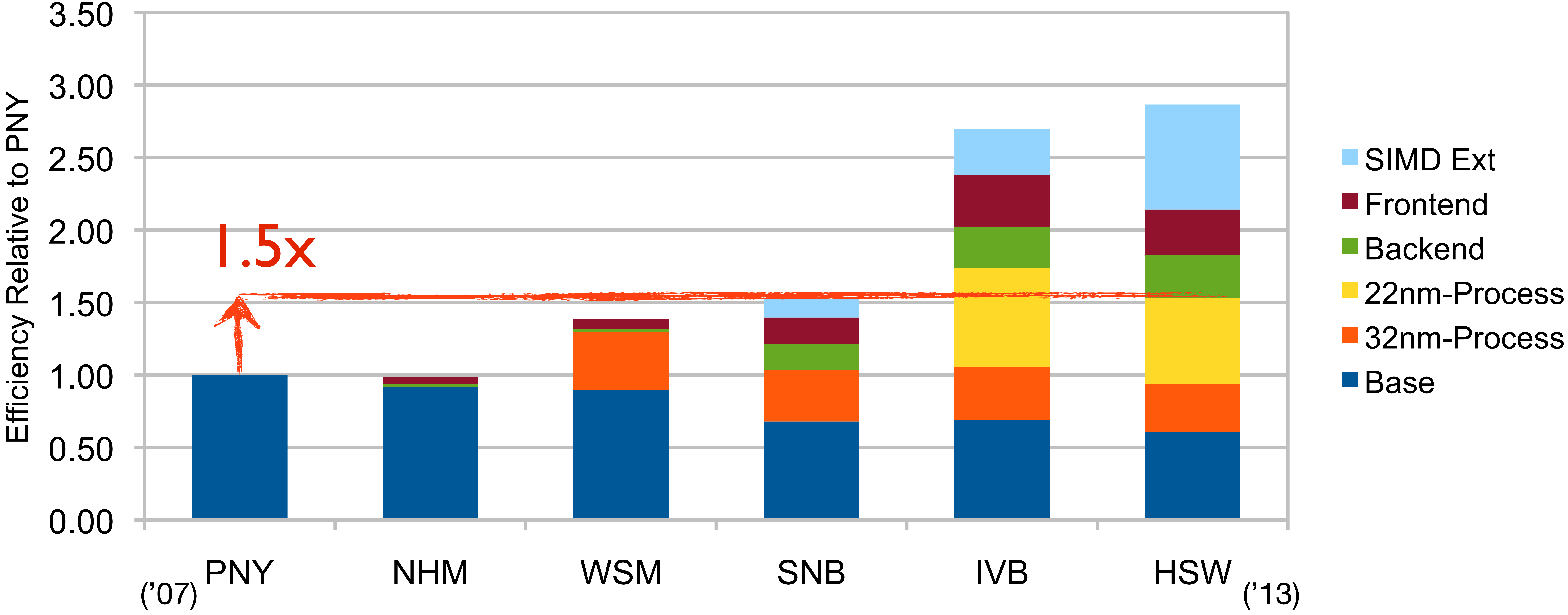
Passive (observational) vs. **Active** (experimental)

Many related ideas!

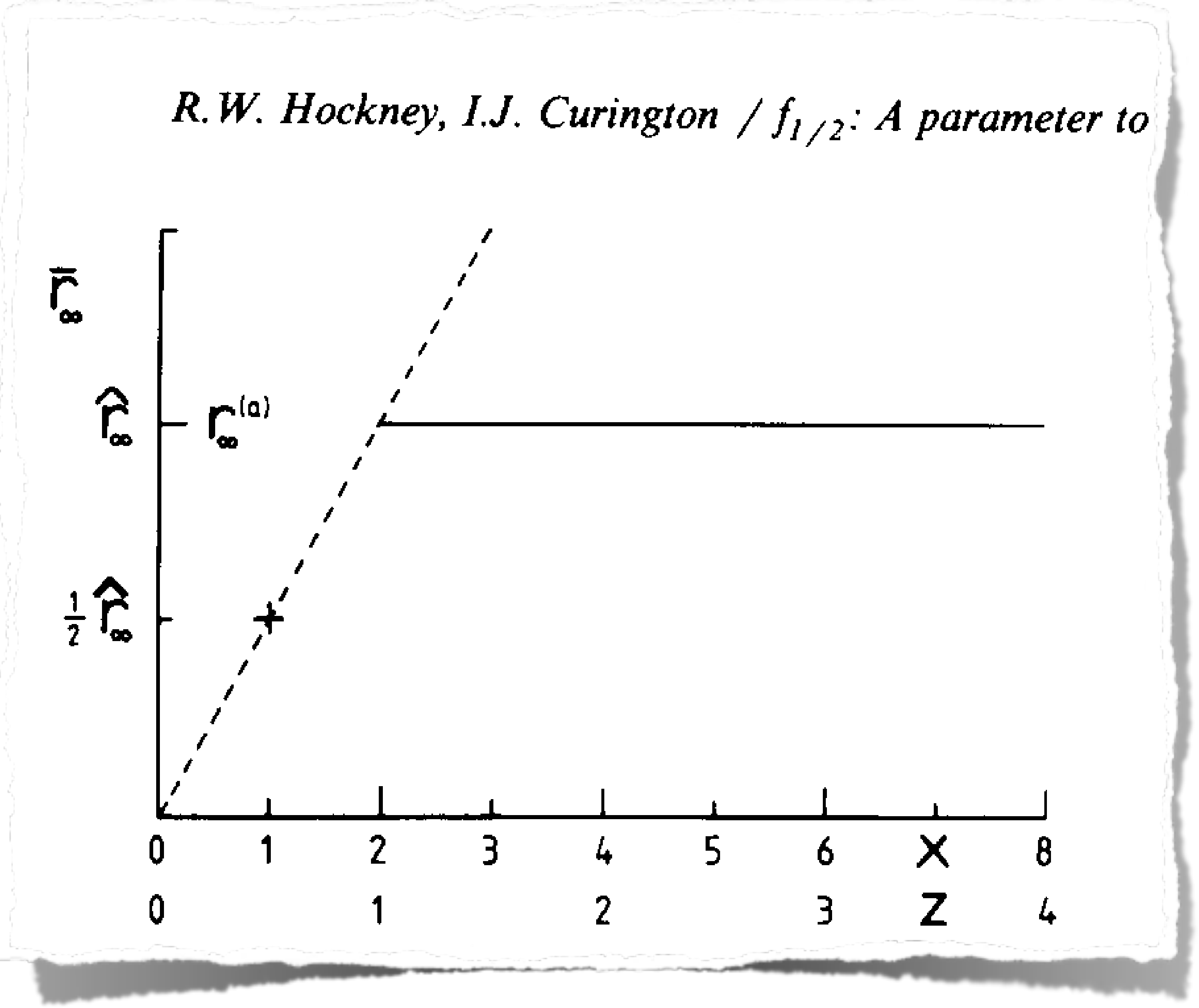
Environmental modifiers: **DVFS**, **Gremlins**

Code modifiers: **autotuning**, **stochastic (super)optimizers**

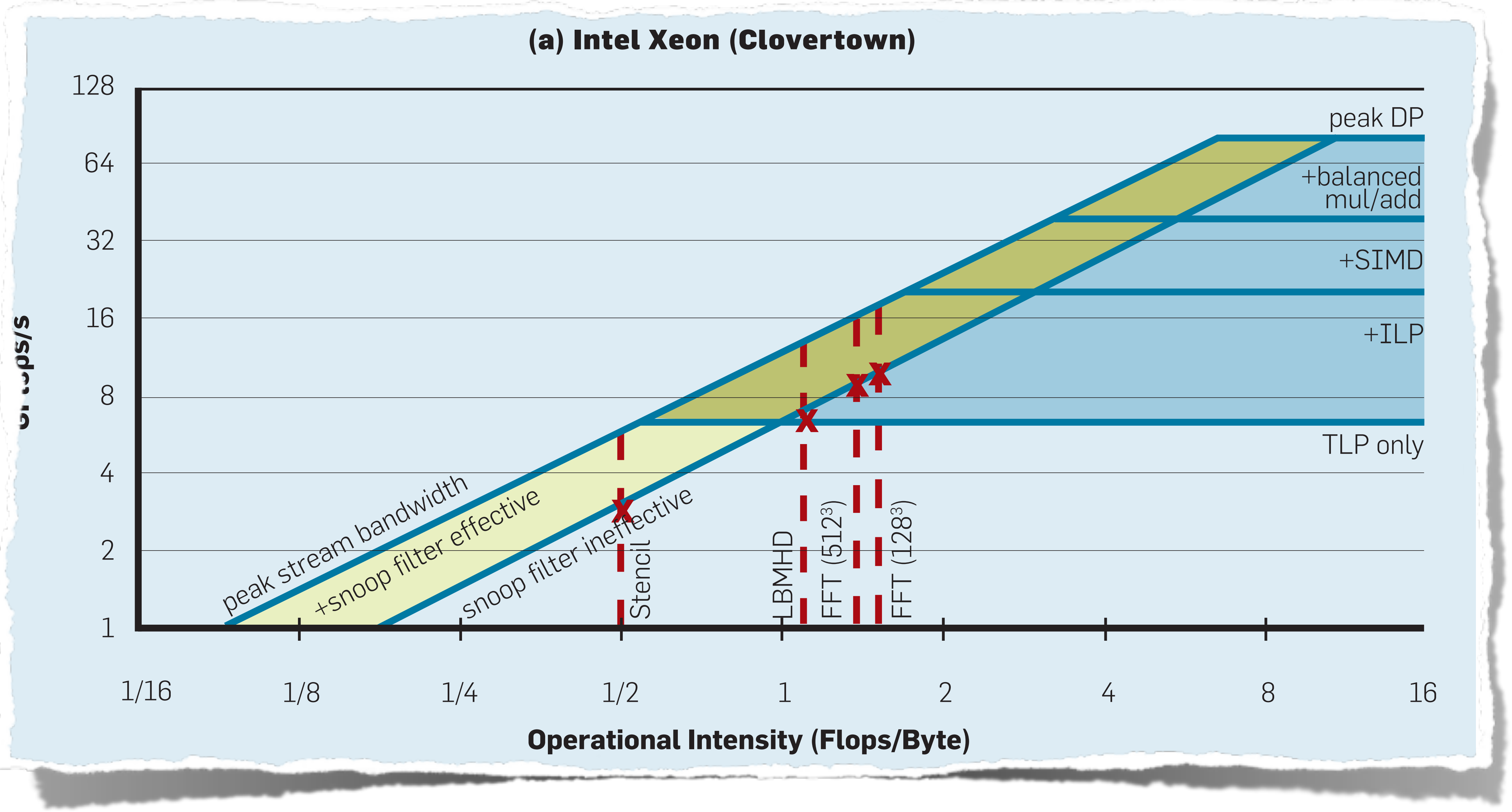
Improvement in Energy Efficiency
Livermore Loops



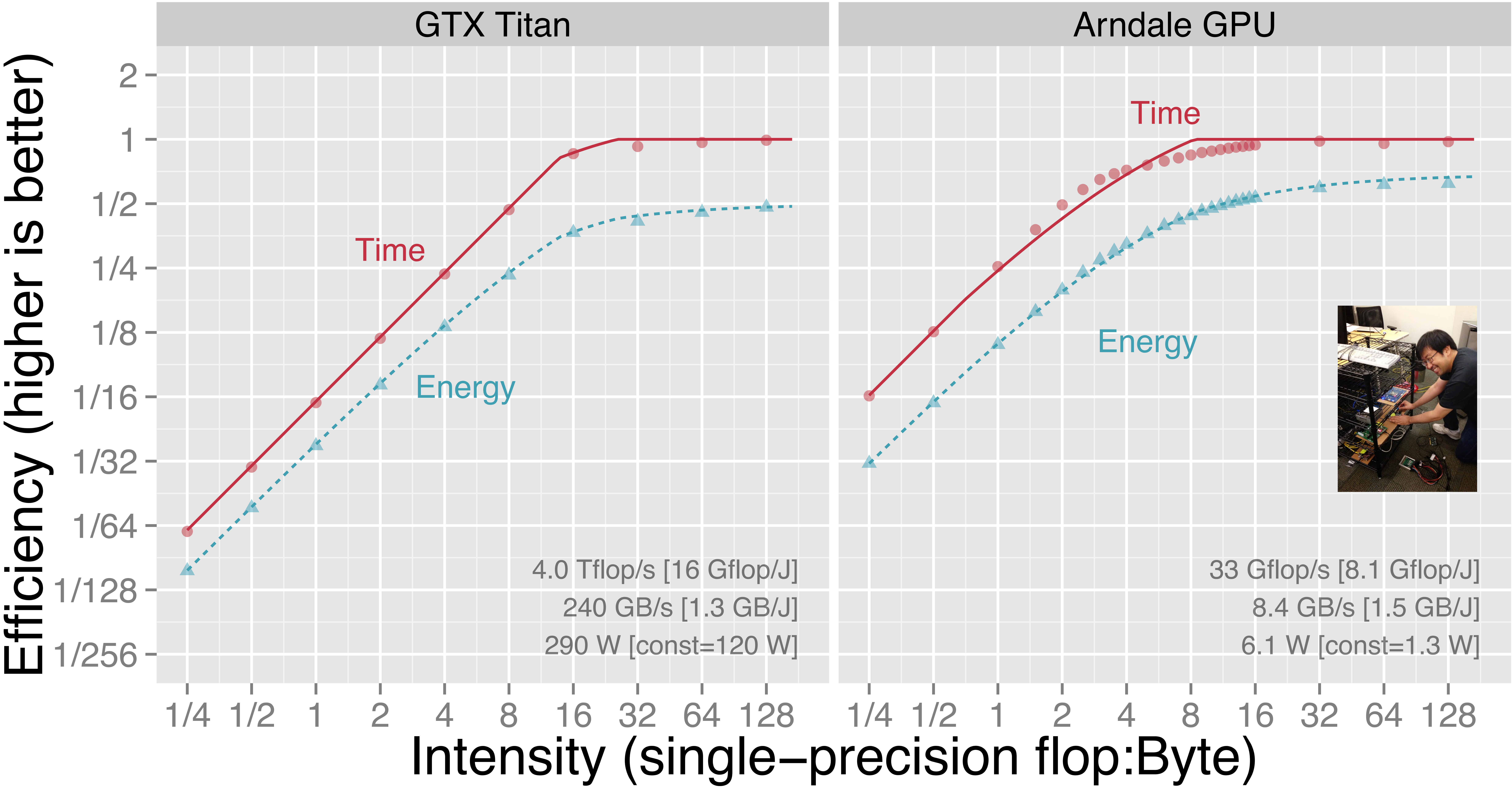
K. Czechowski et al. "Improving the energy-efficiency of big cores." In ISCA'14.



R.W. Hockney and I.J. Curington (1989). “$f_{½}$: A parameter to characterize memory and communication bottlenecks.”
doi: 10.1016/0167-8191(89)90100-2



S. Williams, A. Waterman, D. Patterson (2009).
“Roofline: An insightful visual performance model for multicore architectures.”
doi: 10.1145/1498765.1498785



“Desktop GPU” (NVIDIA)

GTX Titan

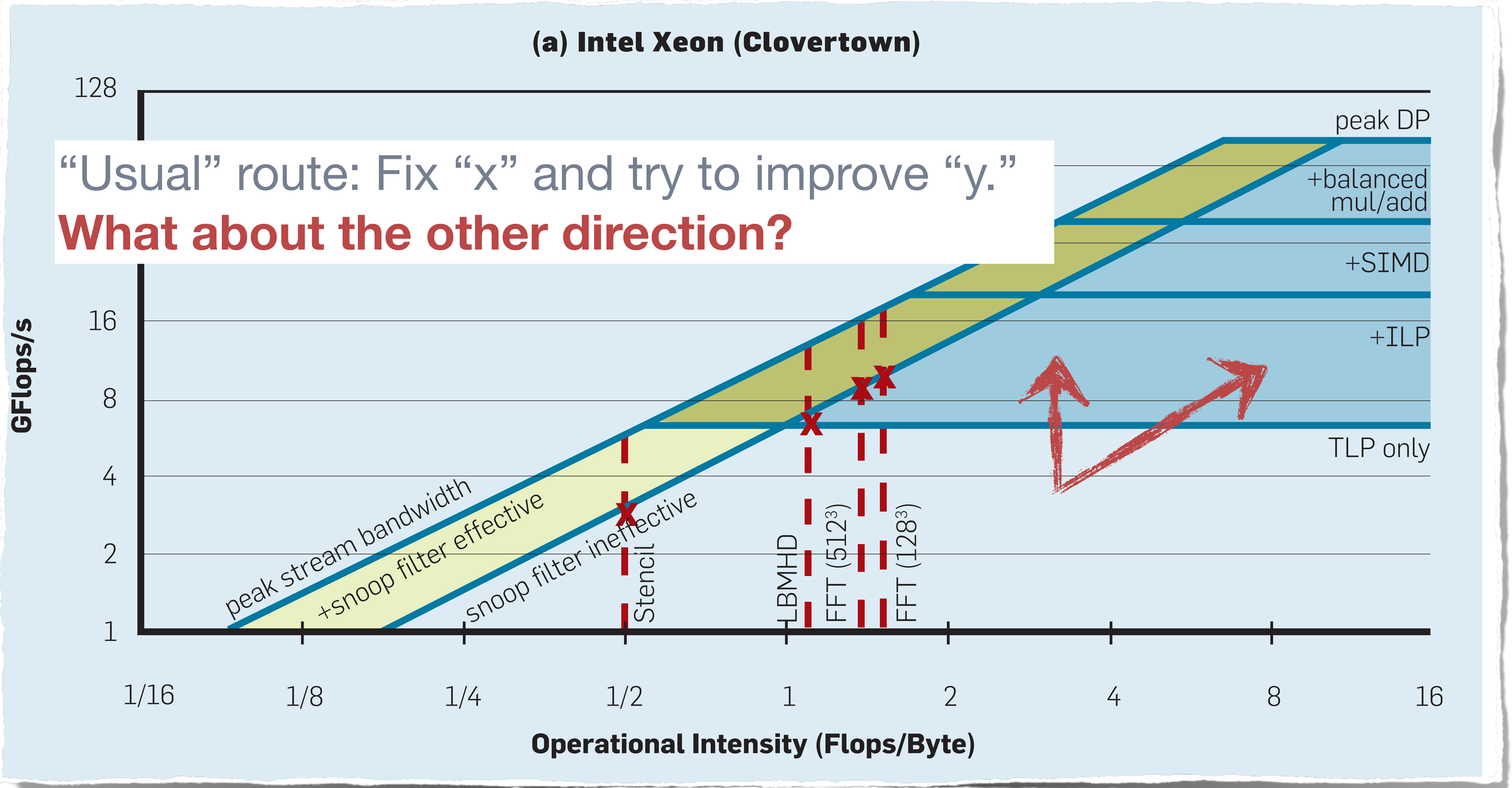
4.0 Tflop/s [16 Gflop/J]
240 GB/s [1.3 GB/J]
290 W [const=120 W]

“Mobile GPU” (Samsung/ARM)

Arndale GPU

33 Gflop/s [8.1 Gflop/J]
8.4 GB/s [1.5 GB/J]
6.1 W [const=1.3 W]

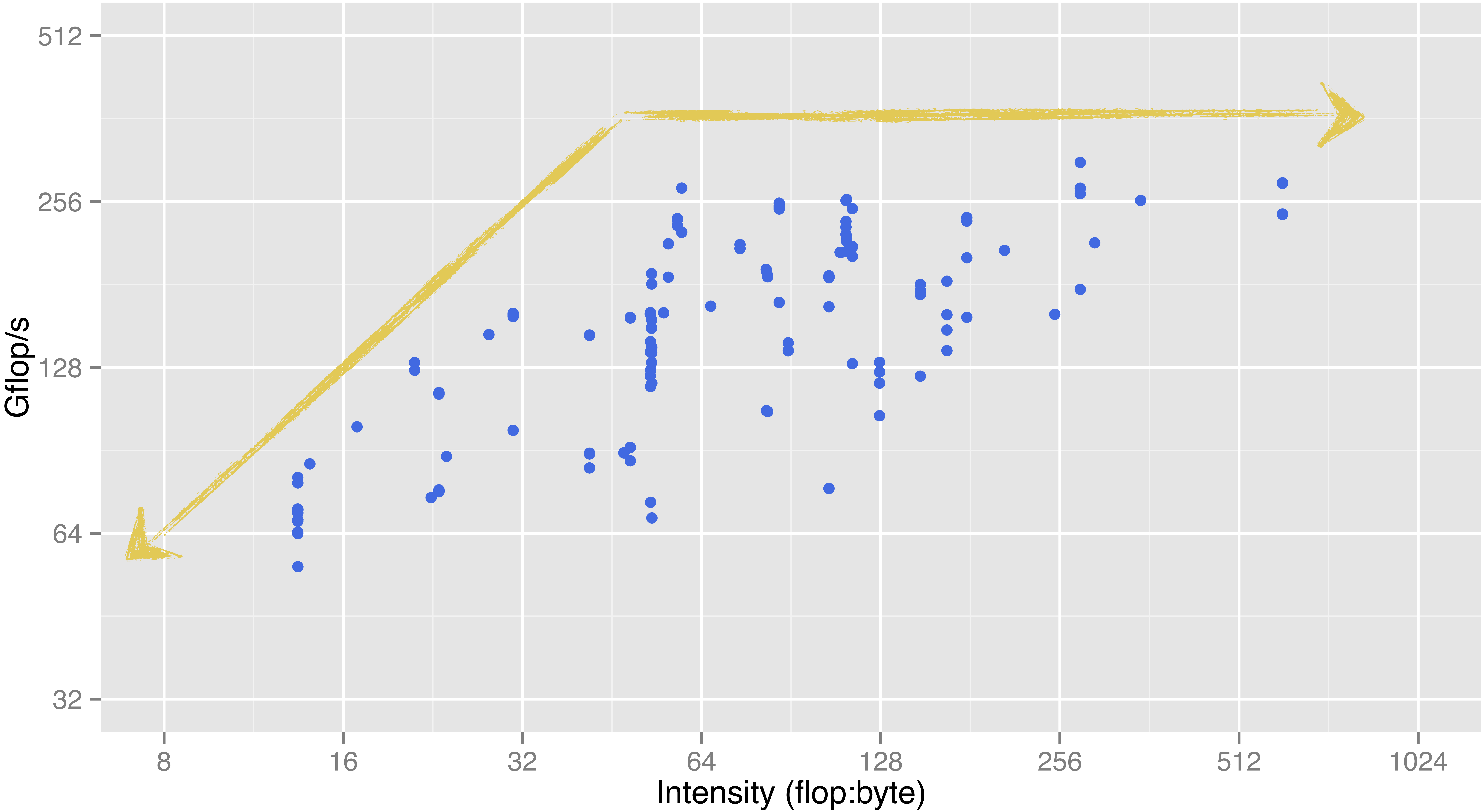
J. Choi et al. (IPDPS’14)



"Usual" route: Fix "x" and try to improve "y."

What about the other direction?

S. Williams, A. Waterman, D. Patterson (CACM'09)



512
256
128
64
32
Gflop/s
8
16
32
64
128
256
512
1024
Intensity (flop:byte)

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

```
forall $v \in V$ do
  $label[v] \leftarrow int(v)$

while … do
  forall $v \in V$ do
    forall $(u, v) \in E$ do
      if $label[u] < label[v]$ then
        $label[u] \leftarrow label[v]$
```

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
Branch−based
Branch−based
Branch−based
Branch−based
Branch−based
Branch−based
Branch−based
Branch−based
Branch−based
Branch−based
Predicted values:
Cache.references
Cache.misses
Branches
Mispredictions

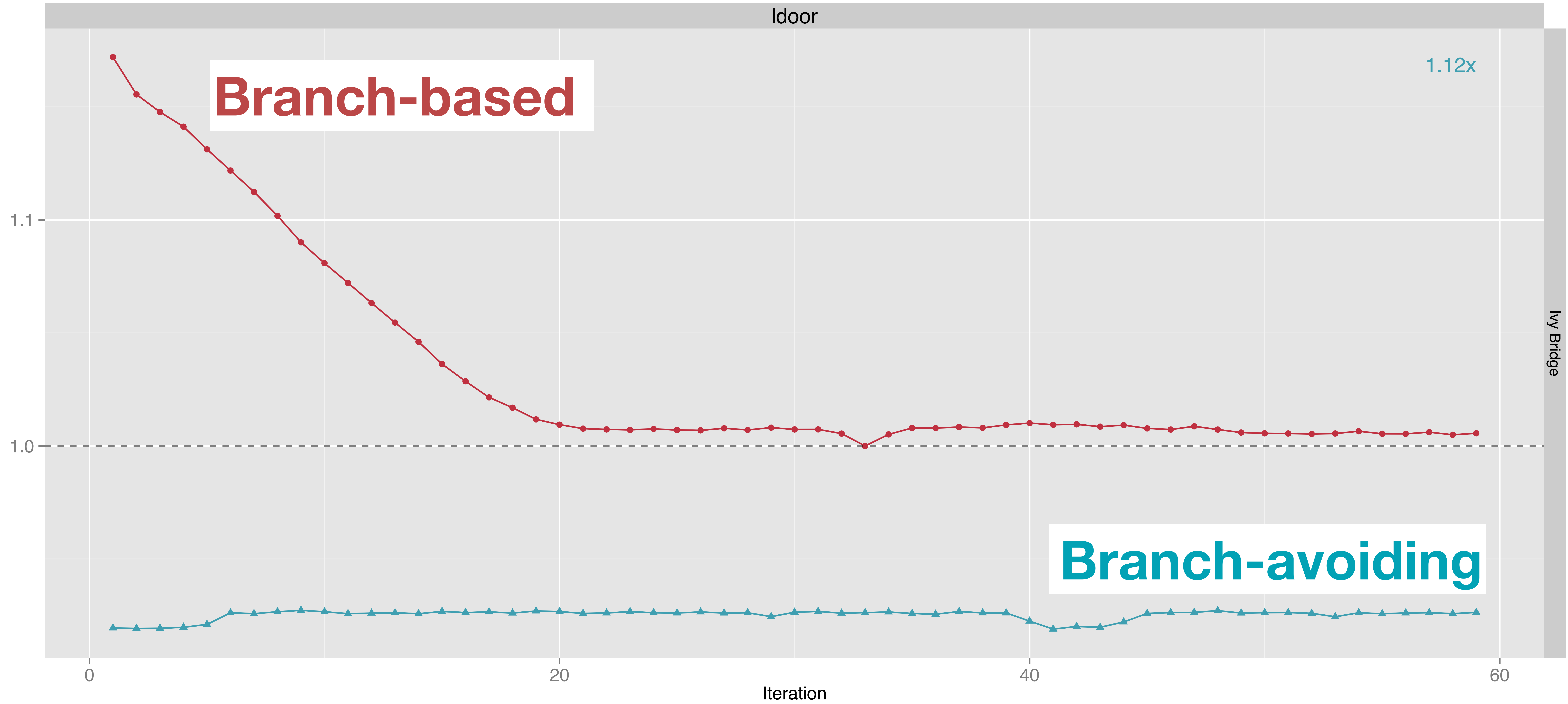
Measured

Modeled

(counters + lasso regression)

Shiloach–Vishkin Connected Components: Cycles

[Normalized to branch–based minimum]

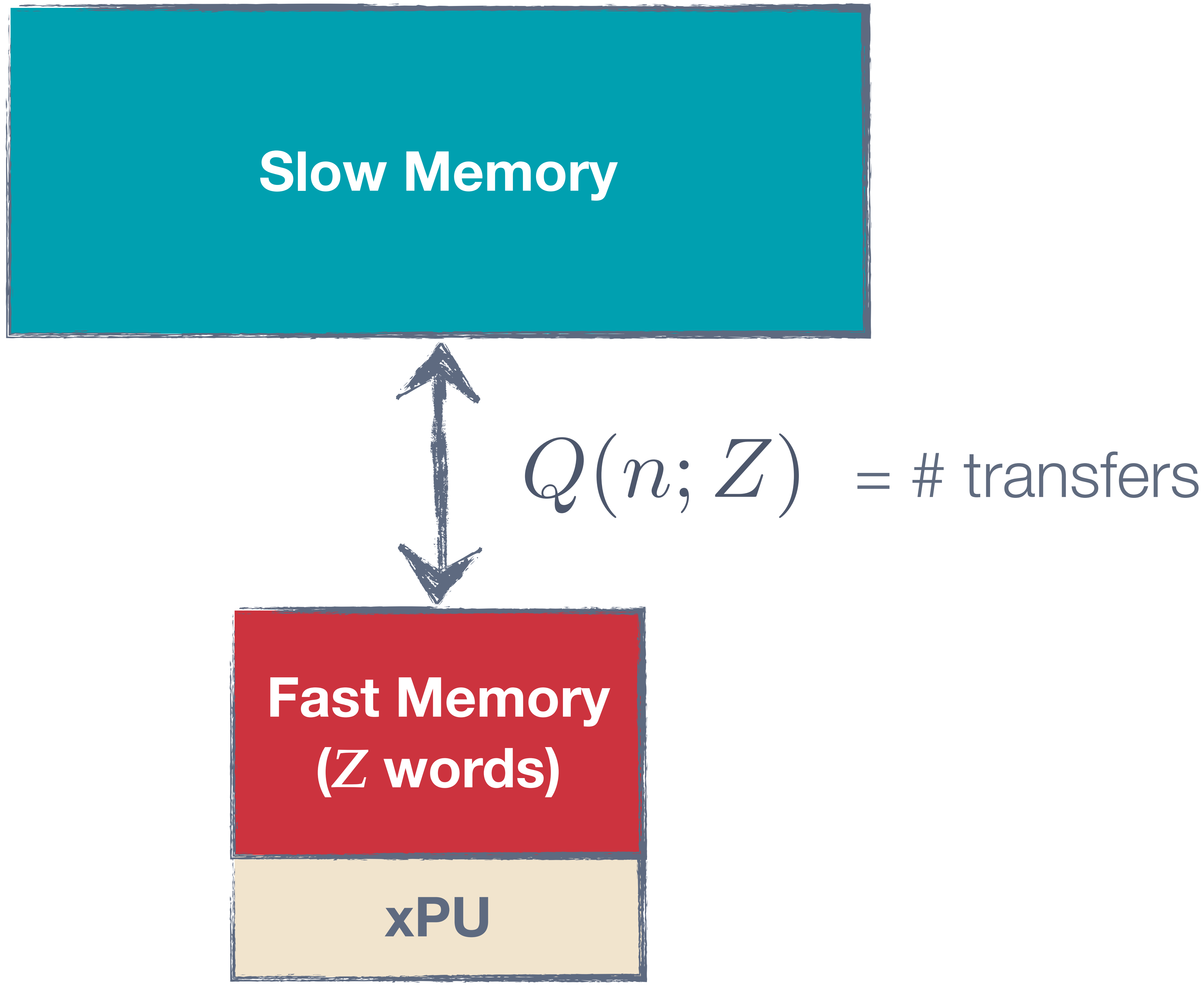


O. Green, M. Dukhan, R. Vuduc. "Branch-avoiding graph algorithms." In SPAA'15.

A frontier:

Performance upper bounds

V. Elango, F. Rastello, L.-N. Pouchet, J. Ramanujam, P. Sadayappan. "On Characterizing the Data Movement Complexity of Computational DAGs for Parallel Execution." In SPAA'14.



Slow Memory

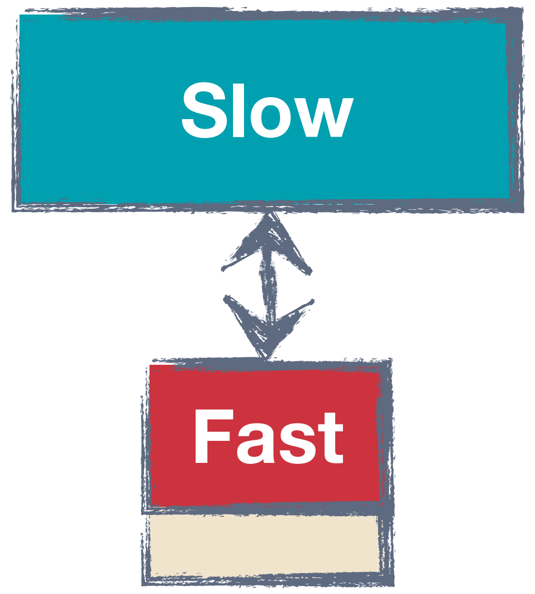
$Q(n;Z)$ = # transfers

Fast Memory (Z words)

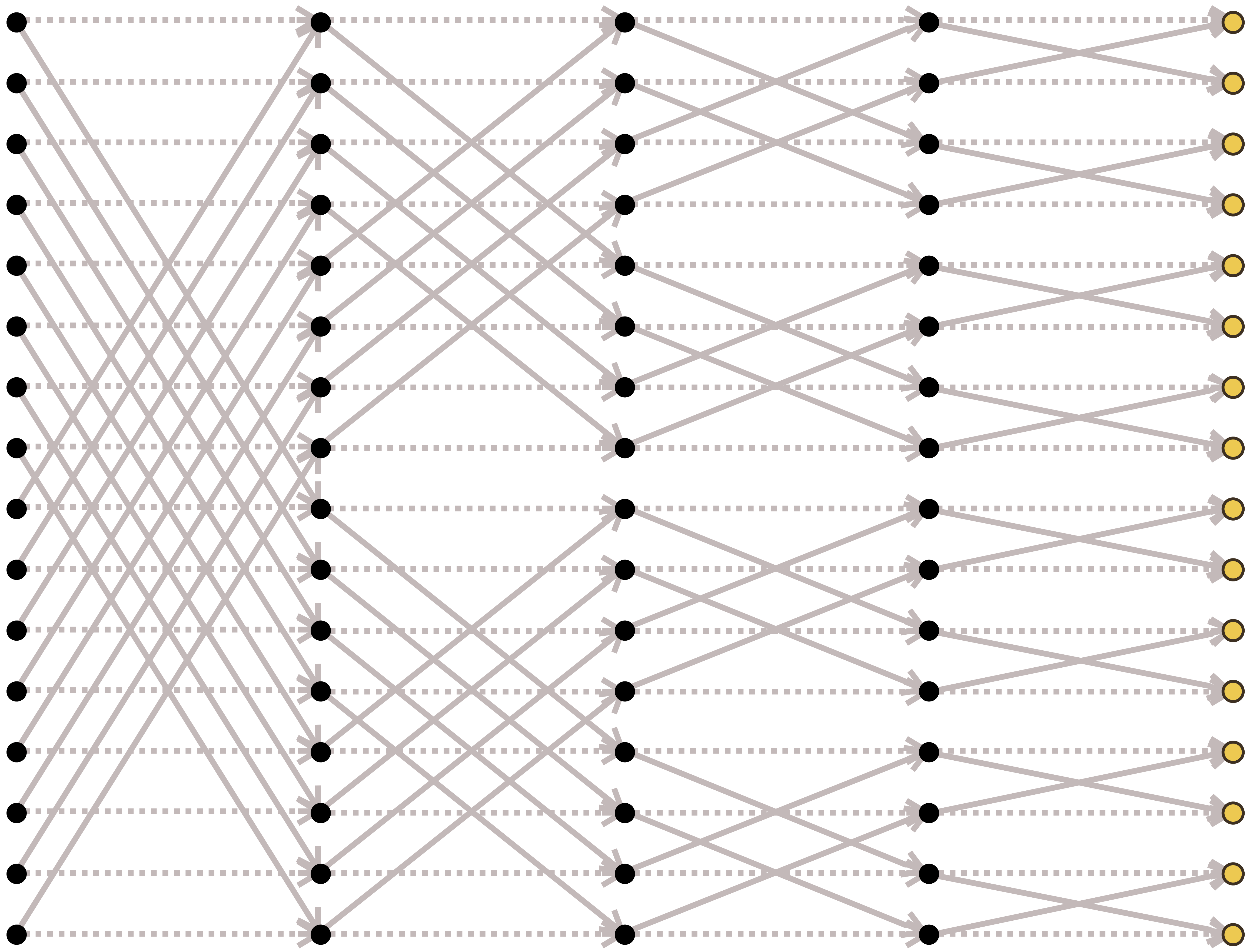
xPU

Goal of algorithm analysis is to estimate or (lower) bound on Q

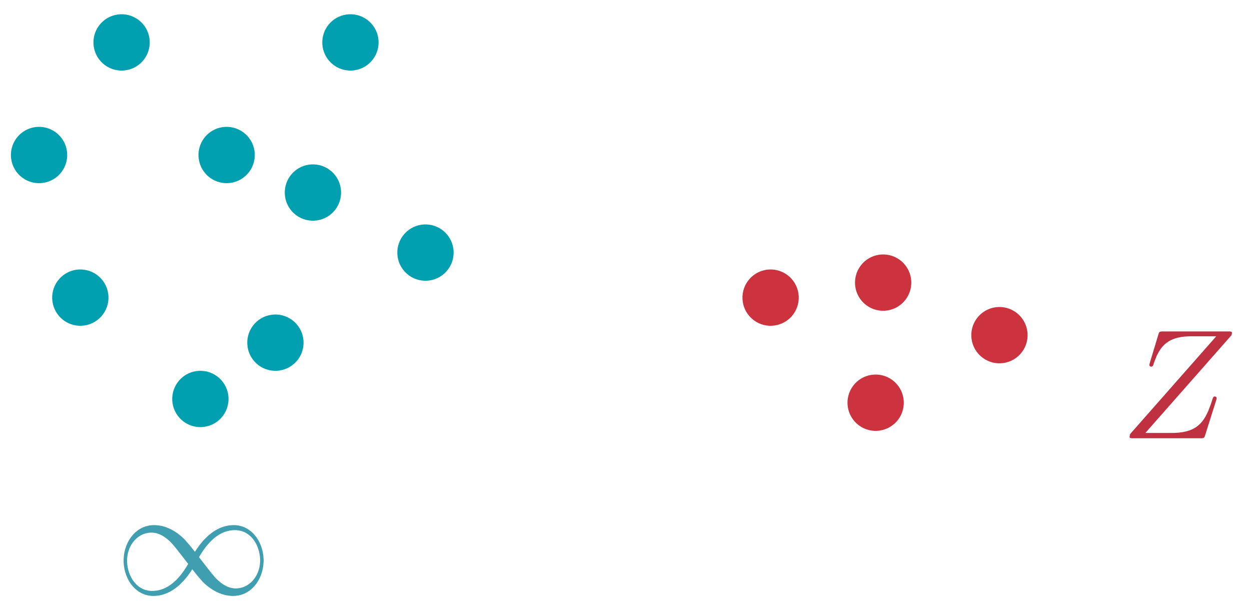
Lower-bounds on Q: Red-blue pebble games



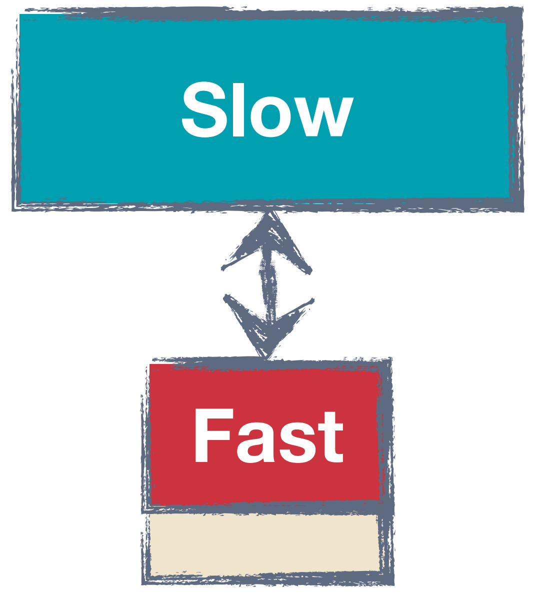
Inputs

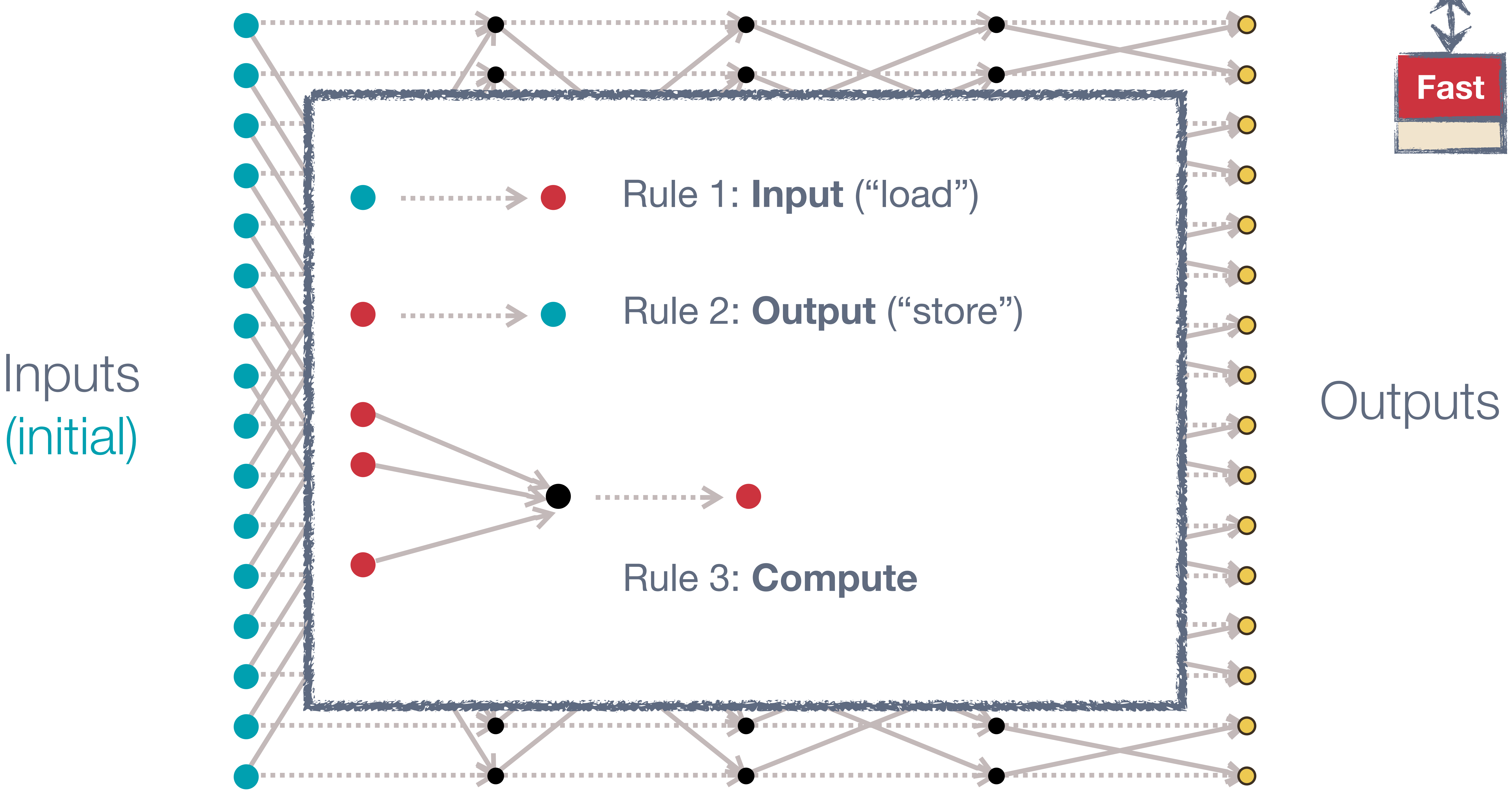


Outputs



Lower-bounds on Q: Red-blue pebble games

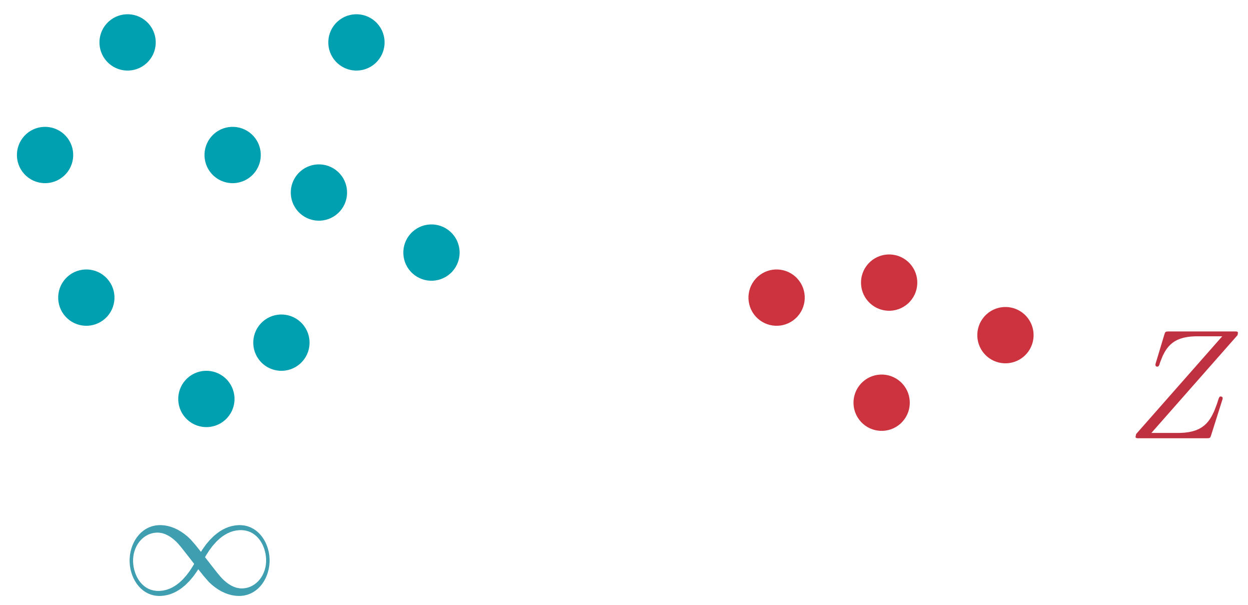




Rule 1: **Input** (“load”)

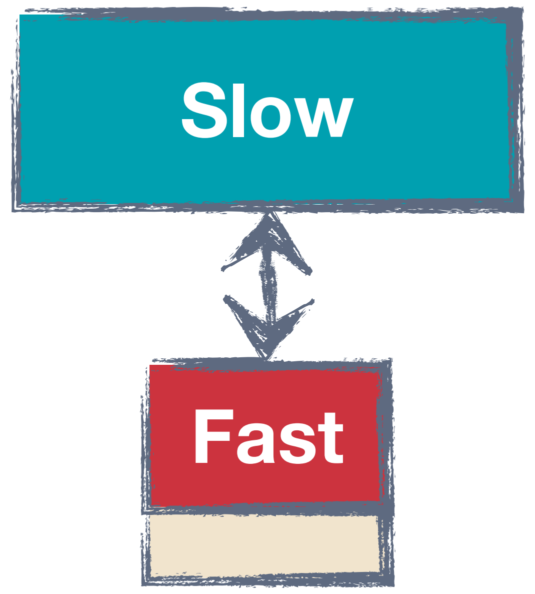
Rule 2: **Output** (“store”)

Rule 3: **Compute**

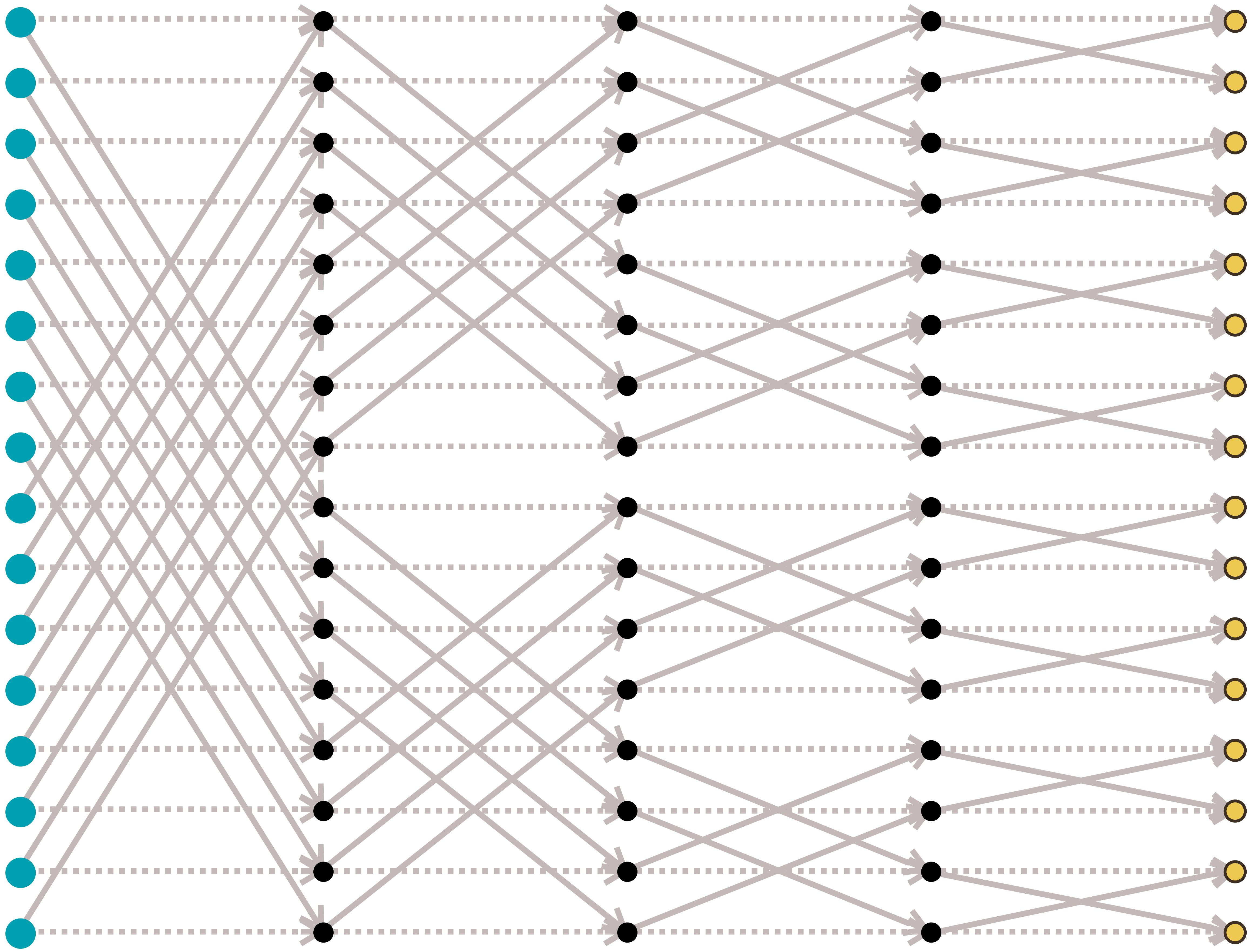


Minimum I/Os (rules 1 & 2) needed to place blue pebbles on outputs?

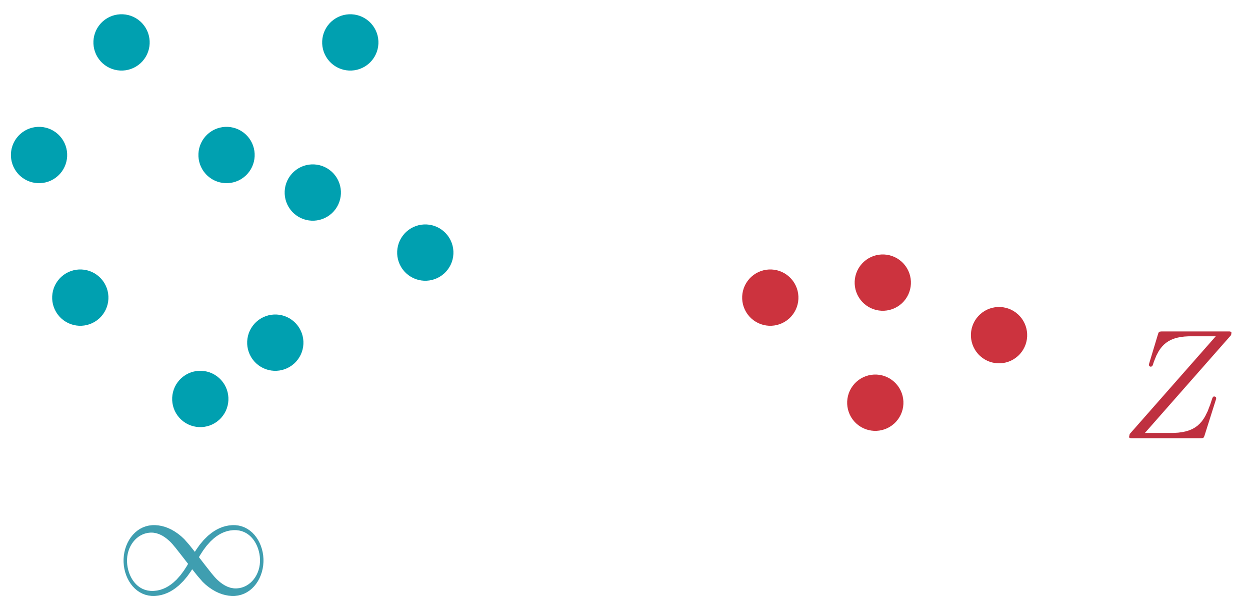
Lower-bounds on Q: Red-blue pebble games


Slow

Fast


Inputs
(initial)

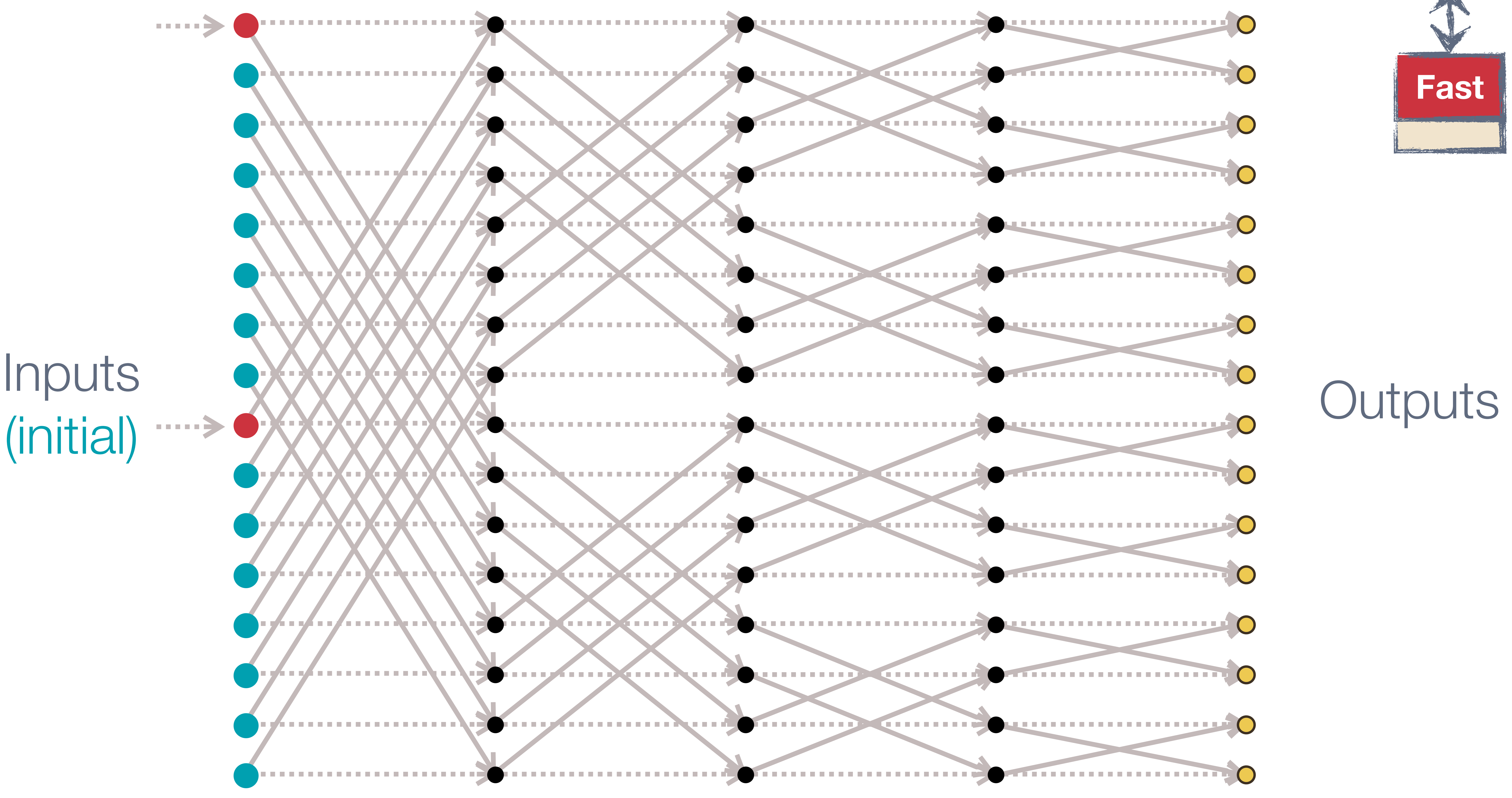
Outputs

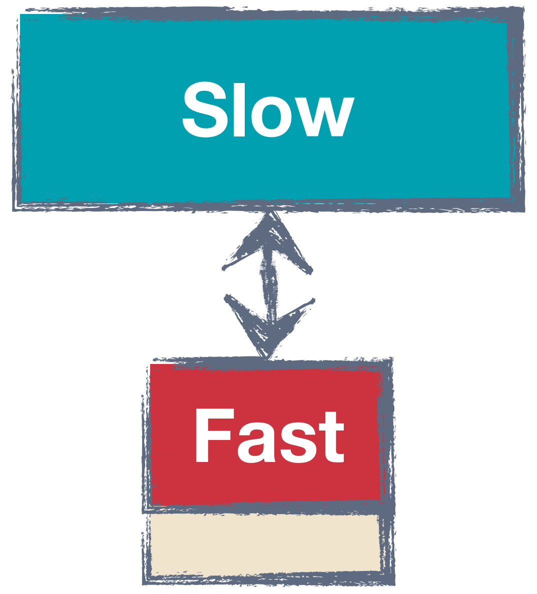

∞

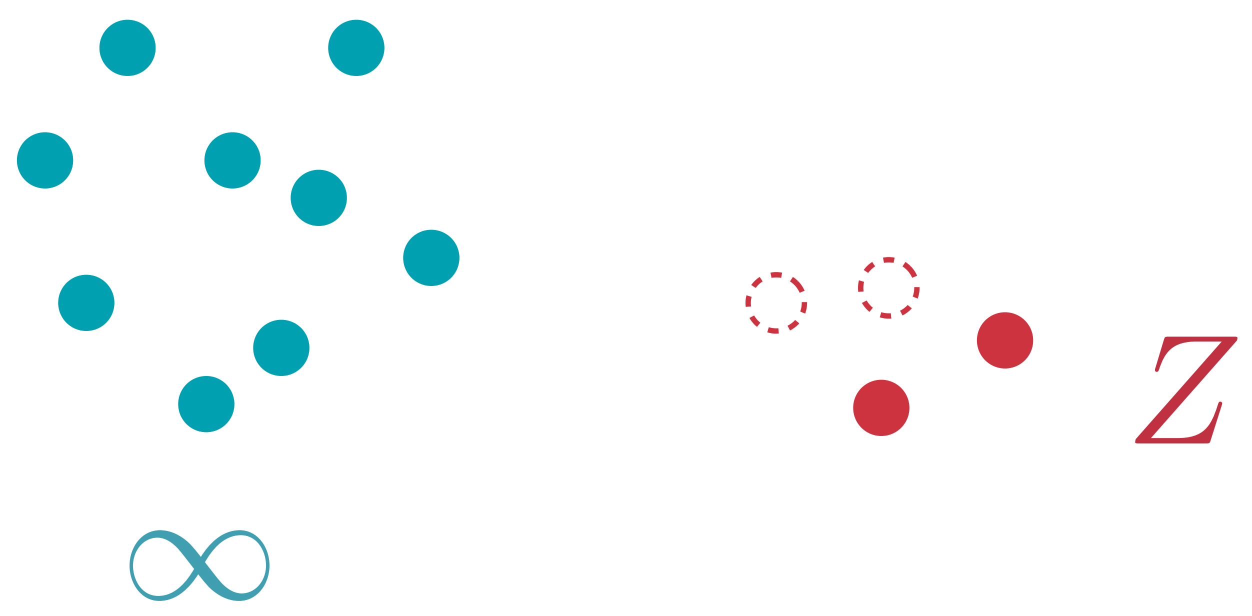
Z

Minimum I/Os (rules 1 & 2) needed to place blue pebbles on outputs?

Lower-bounds on Q: Red-blue pebble games

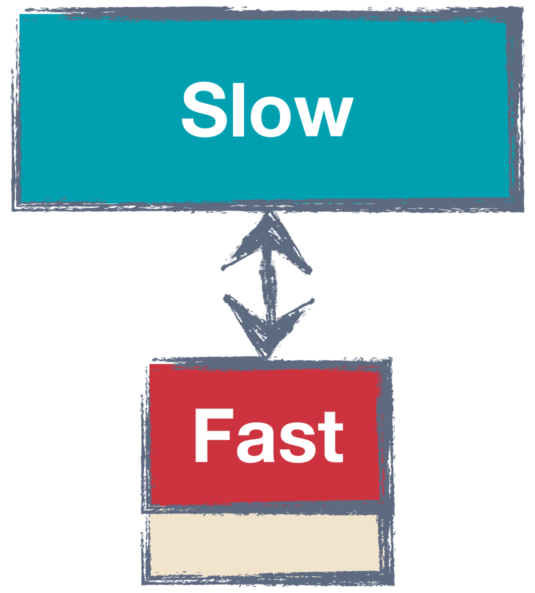


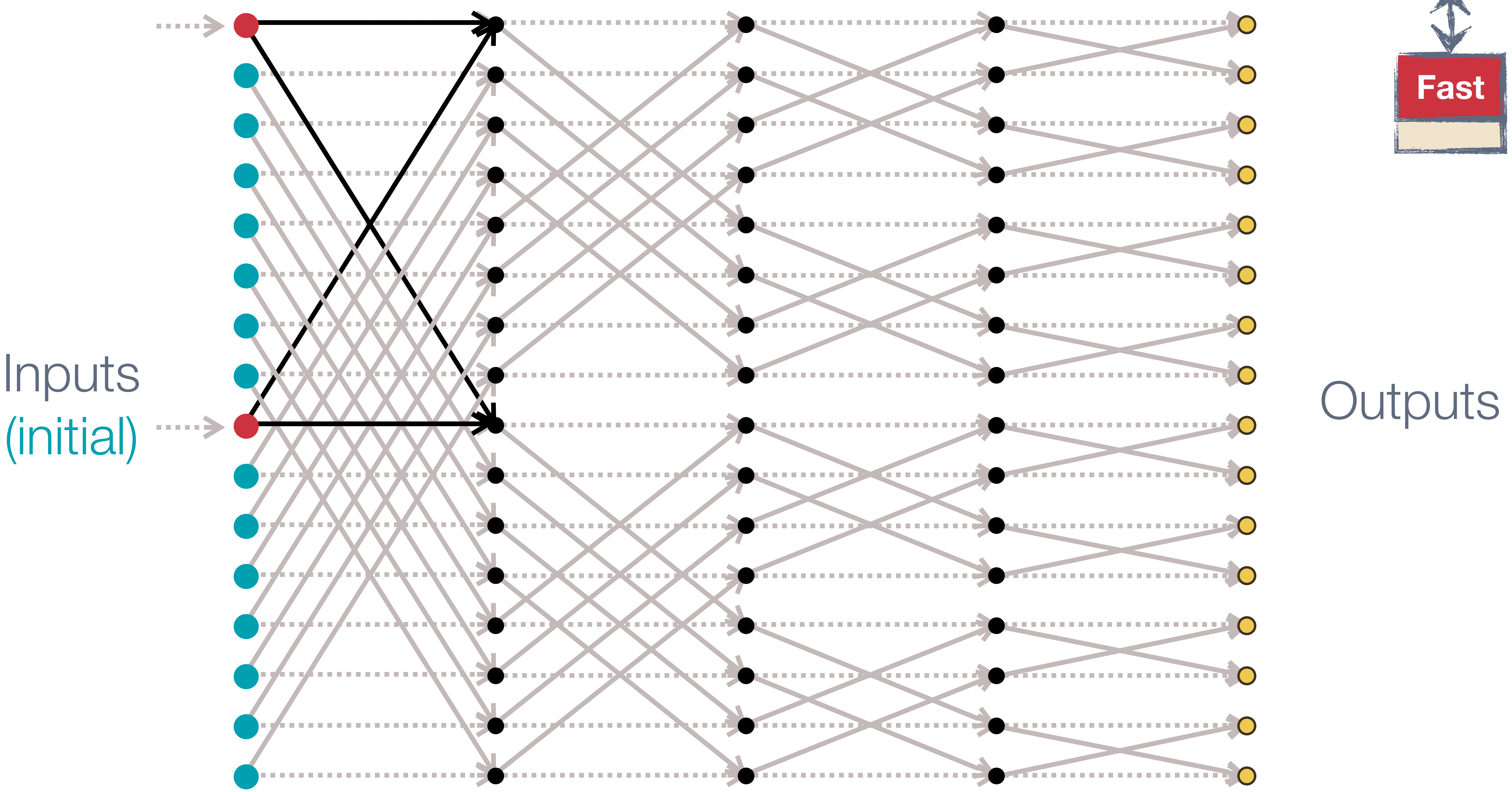


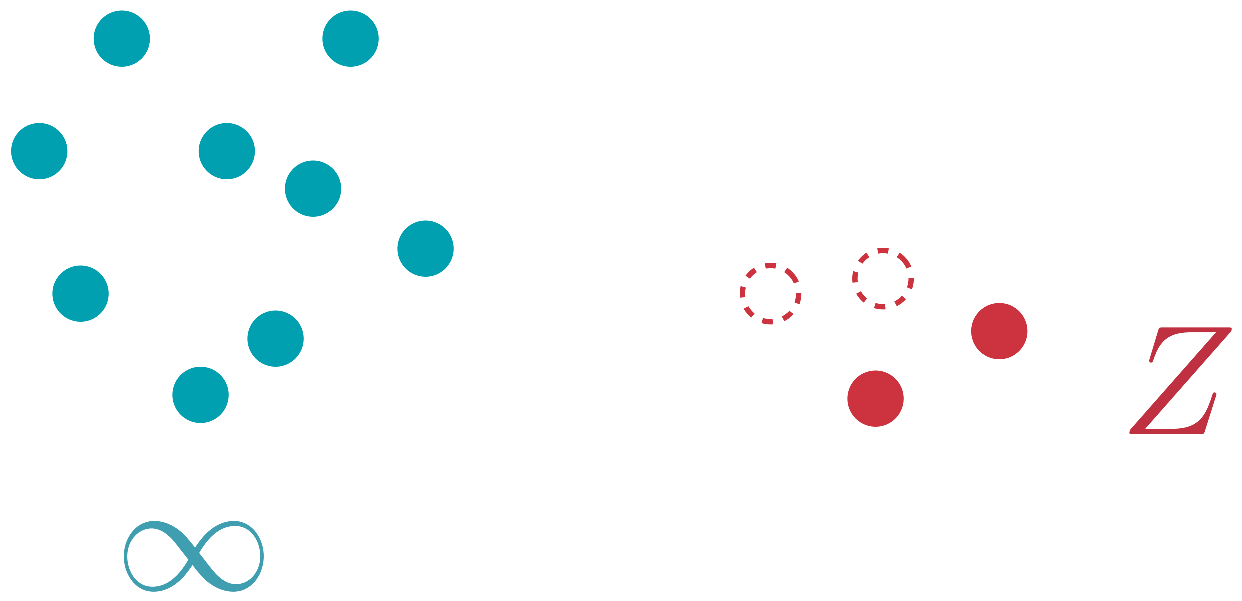


Minimum I/Os (rules 1 & 2) needed to place blue pebbles on outputs?

Lower-bounds on Q: Red-blue pebble games

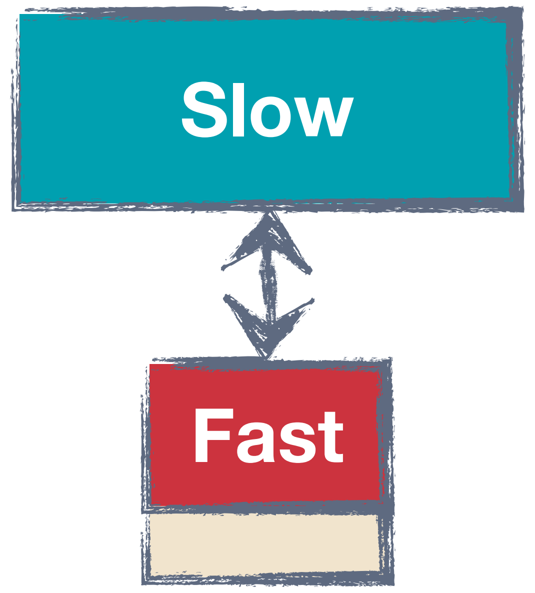


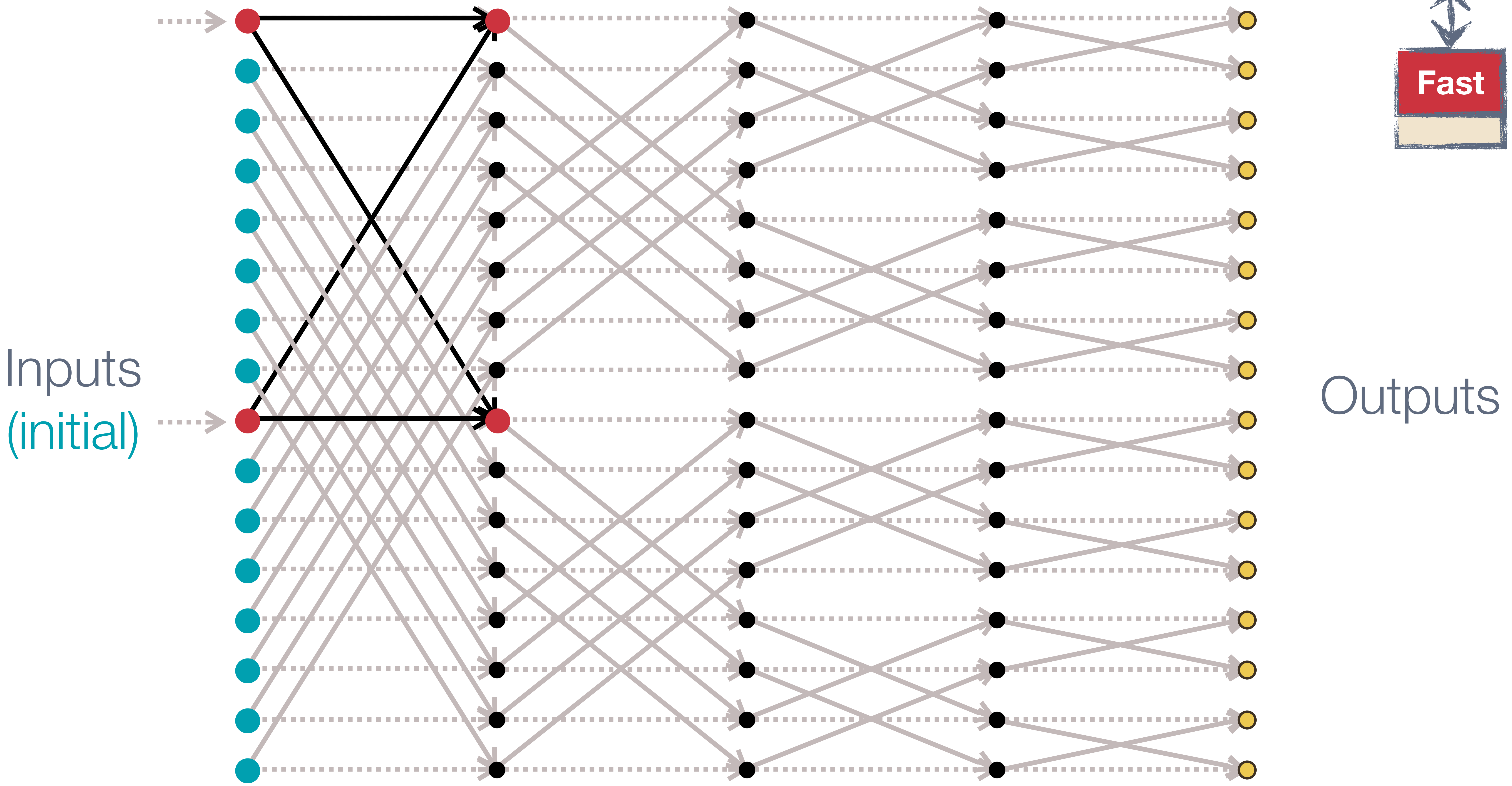


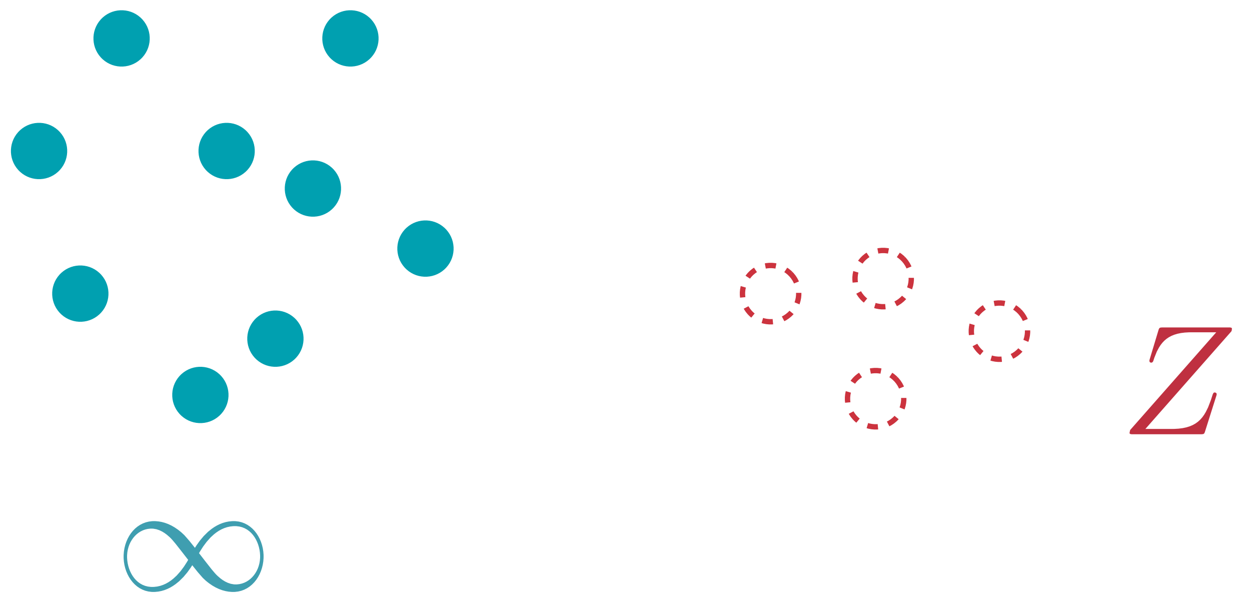


Minimum I/Os (rules 1 & 2) needed to place blue pebbles on outputs?

Lower-bounds on Q: Red-blue pebble games

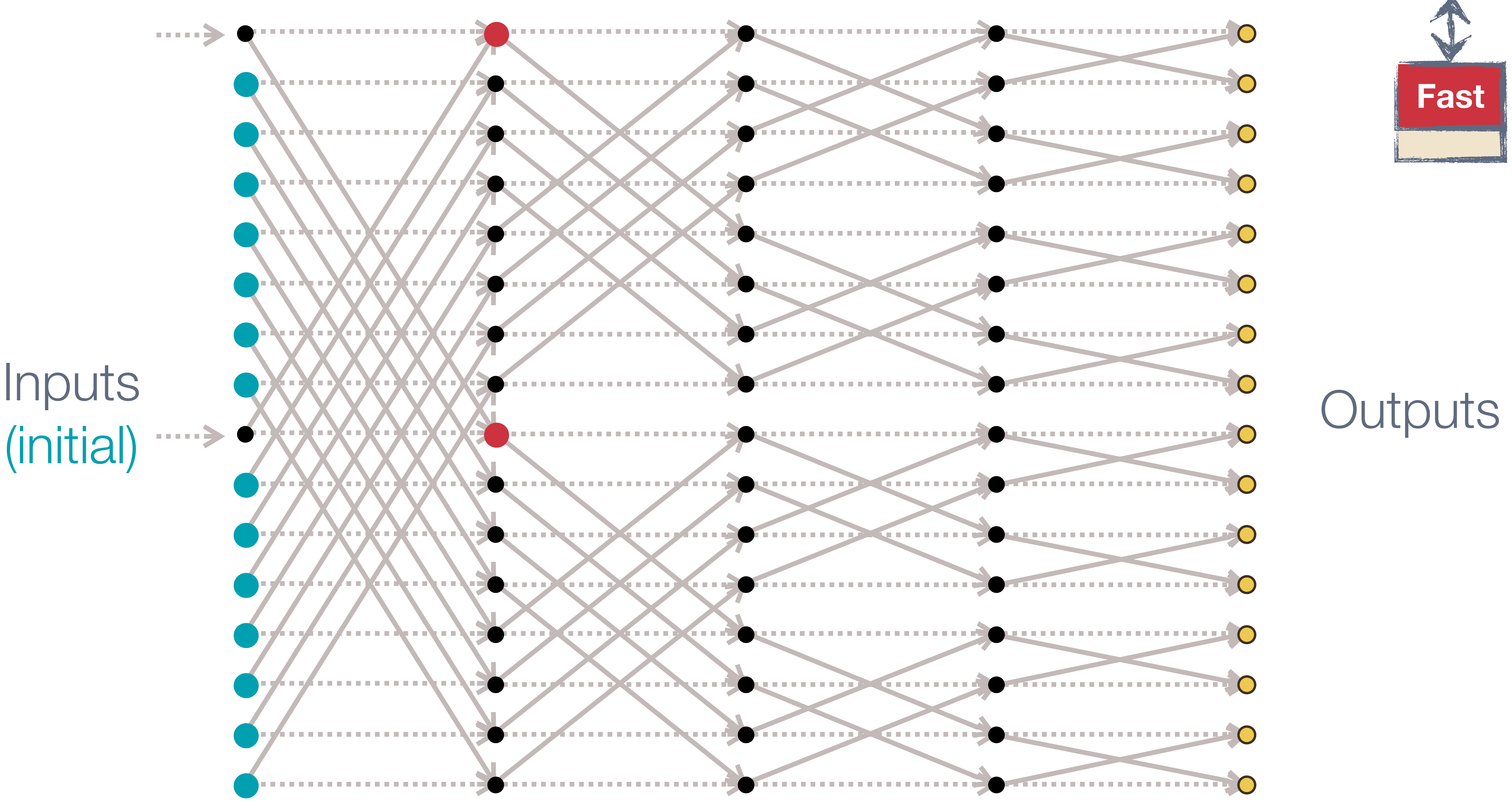


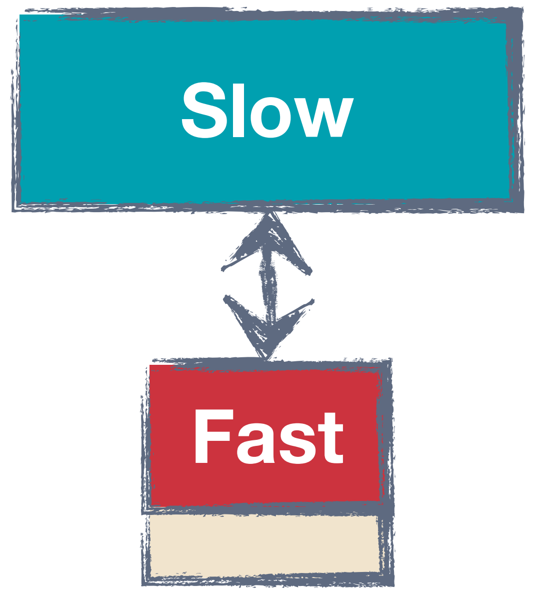


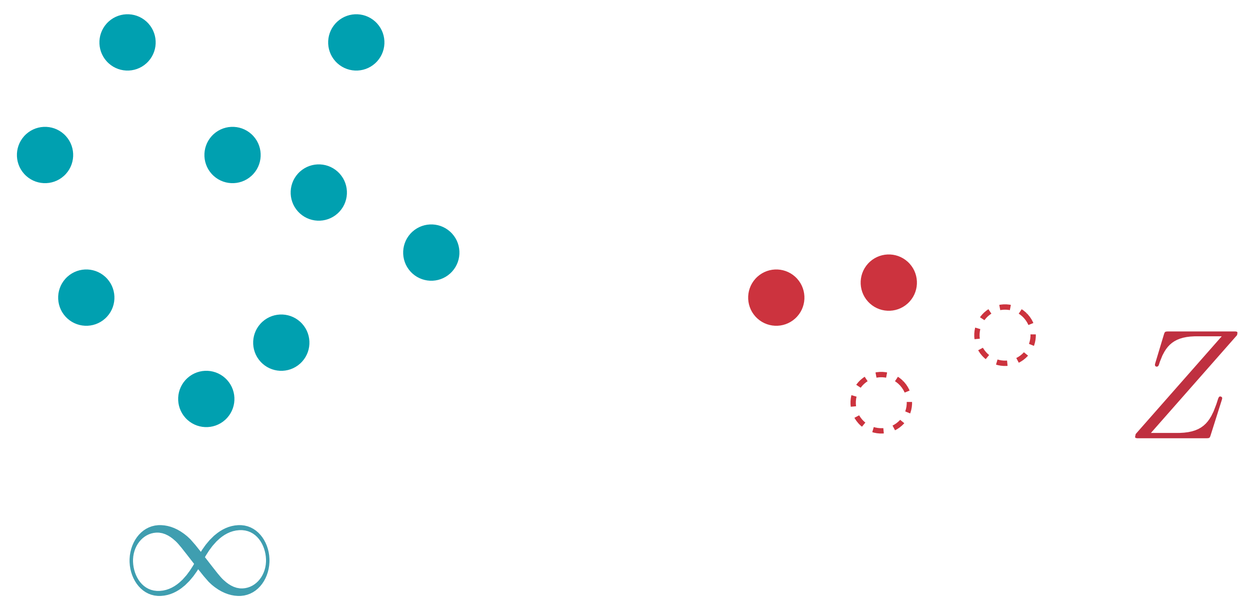


Minimum I/Os (rules 1 & 2) needed to place blue pebbles on outputs?

Lower-bounds on Q: Red-blue pebble games

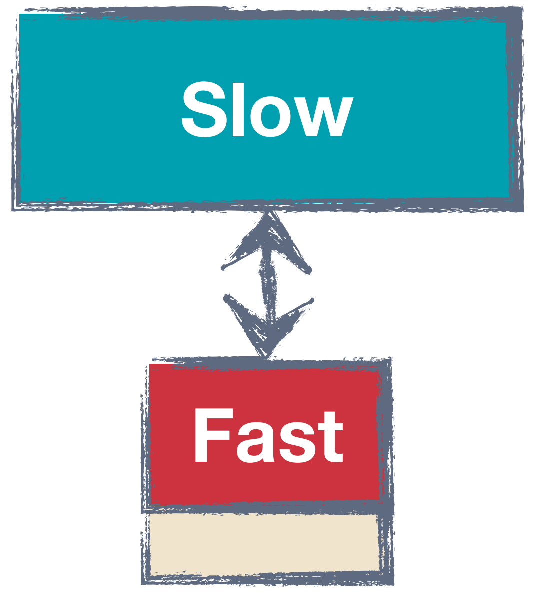


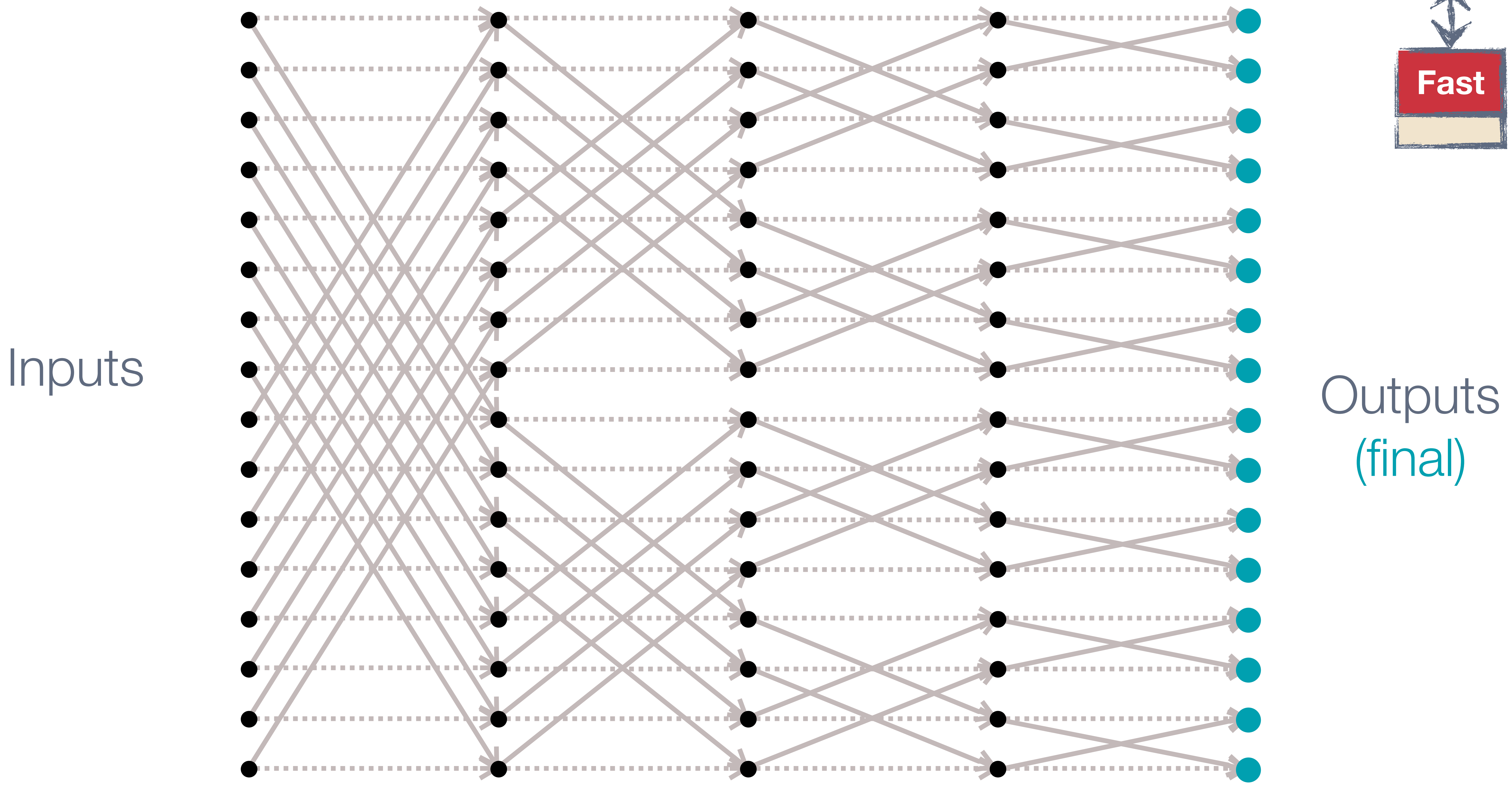


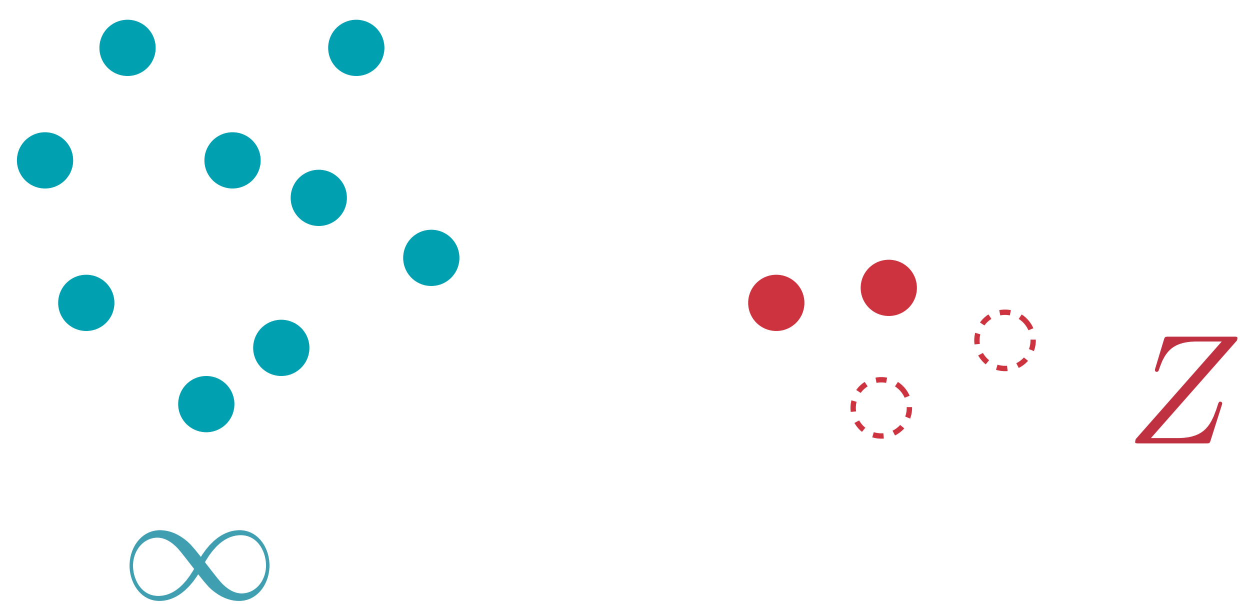


Minimum I/Os (rules 1 & 2) needed to place blue pebbles on outputs?

Lower-bounds on Q: Red-blue pebble games



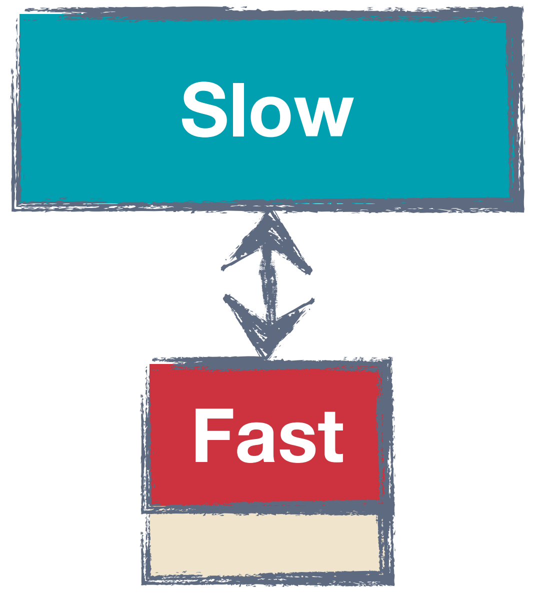


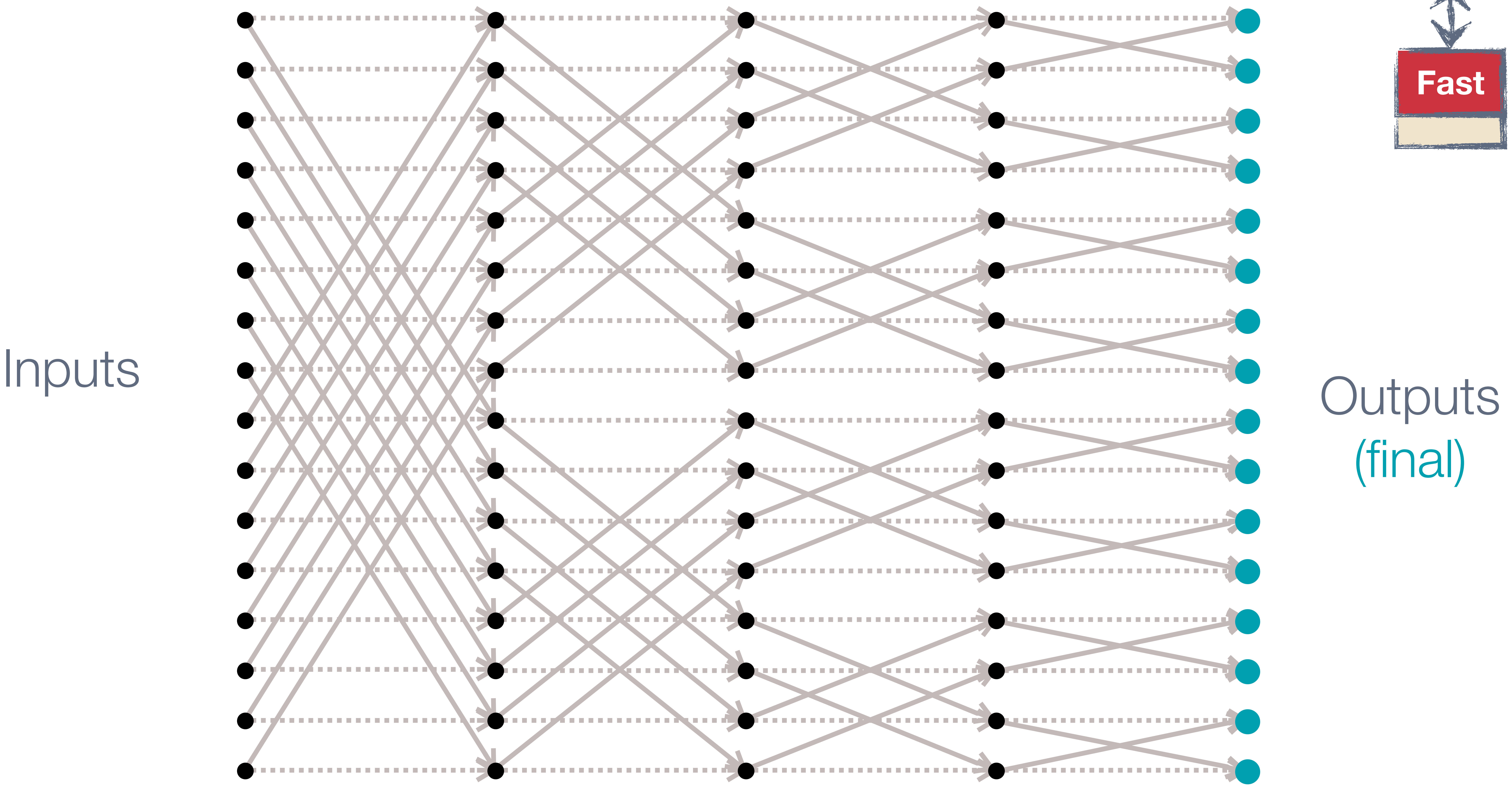


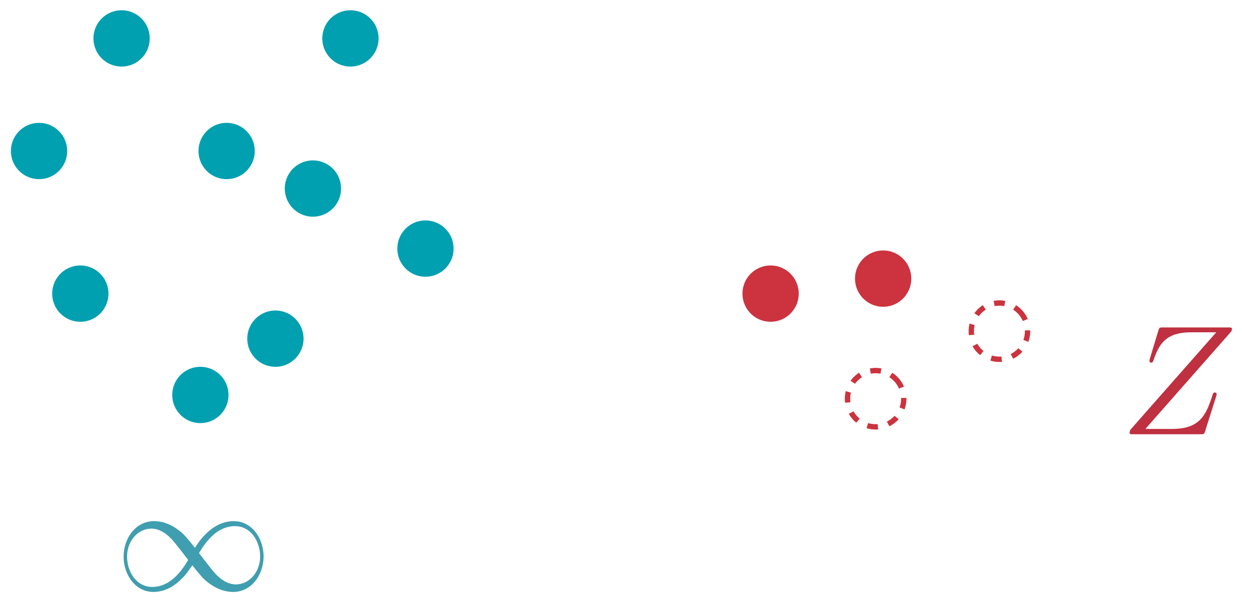
(Hong & Kung '81)

$$Q(n; Z) = \Omega\left(\frac{n \log n}{\log Z}\right)$$

Lower-bounds on Q: Red-blue pebble games





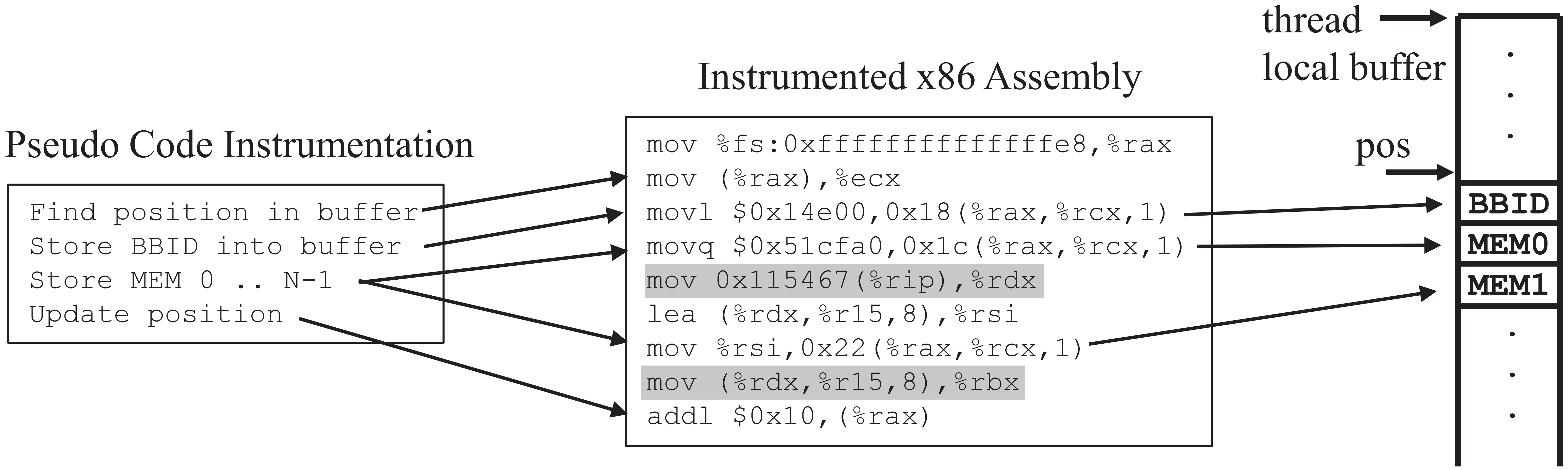


Insight: This representation is computable

V. Elango, F. Rastello, L.-N. Pouchet, J. Ramanujam, P. Sadayappan. "On Characterizing the Data Movement Complexity of Computational DAGs for Parallel Execution." In SPAA'14.

Contech: Efficiently Generating Dynamic Task Graphs for Arbitrary Parallel Programs

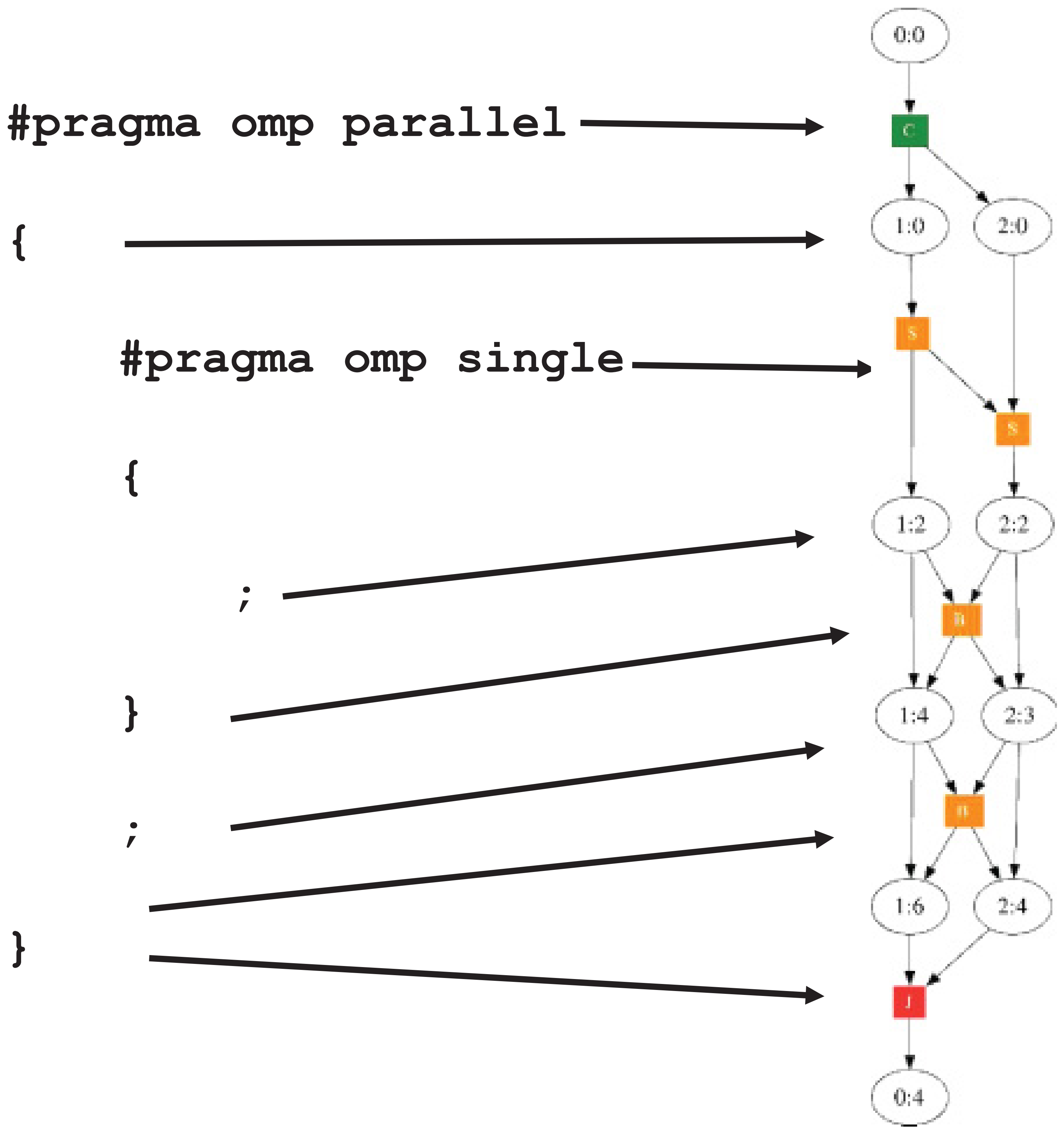
BRIAN P. RAILING, ERIC R. HEIN, and THOMAS M. CONTE, Georgia Institute of Technology



```
Find position in buffer
Store BBID into buffer
Store MEM 0 .. N-1
Update position
```

```
mov %fs:0xffffffffffffffe8,%rax
mov (%rax),%ecx
movl $0x14e00,0x18(%rax,%rcx,1)
movq $0x51cfa0,0x1c(%rax,%rcx,1)
mov 0x115467(%rip),%rdx
lea (%rdx,%r15,8),%rsi
mov %rsi,0x22(%rax,%rcx,1)
mov (%rdx,%r15,8),%rbx
addl $0x10,(%rax)
```

Fig. 5. Runtime instrumentation design (gray instructions are original code).



```
#pragma omp parallel
{
    #pragma omp single
    {
        ;
    }
    ;
}
```

Fig. 4. Simple OpenMP program as a Contech task graph.

Kent Czechowski

Kent's idea:

Pressure point analysis (PPA)

Iteratively rewrite the input program in a controlled fashion, then re-analyze it.

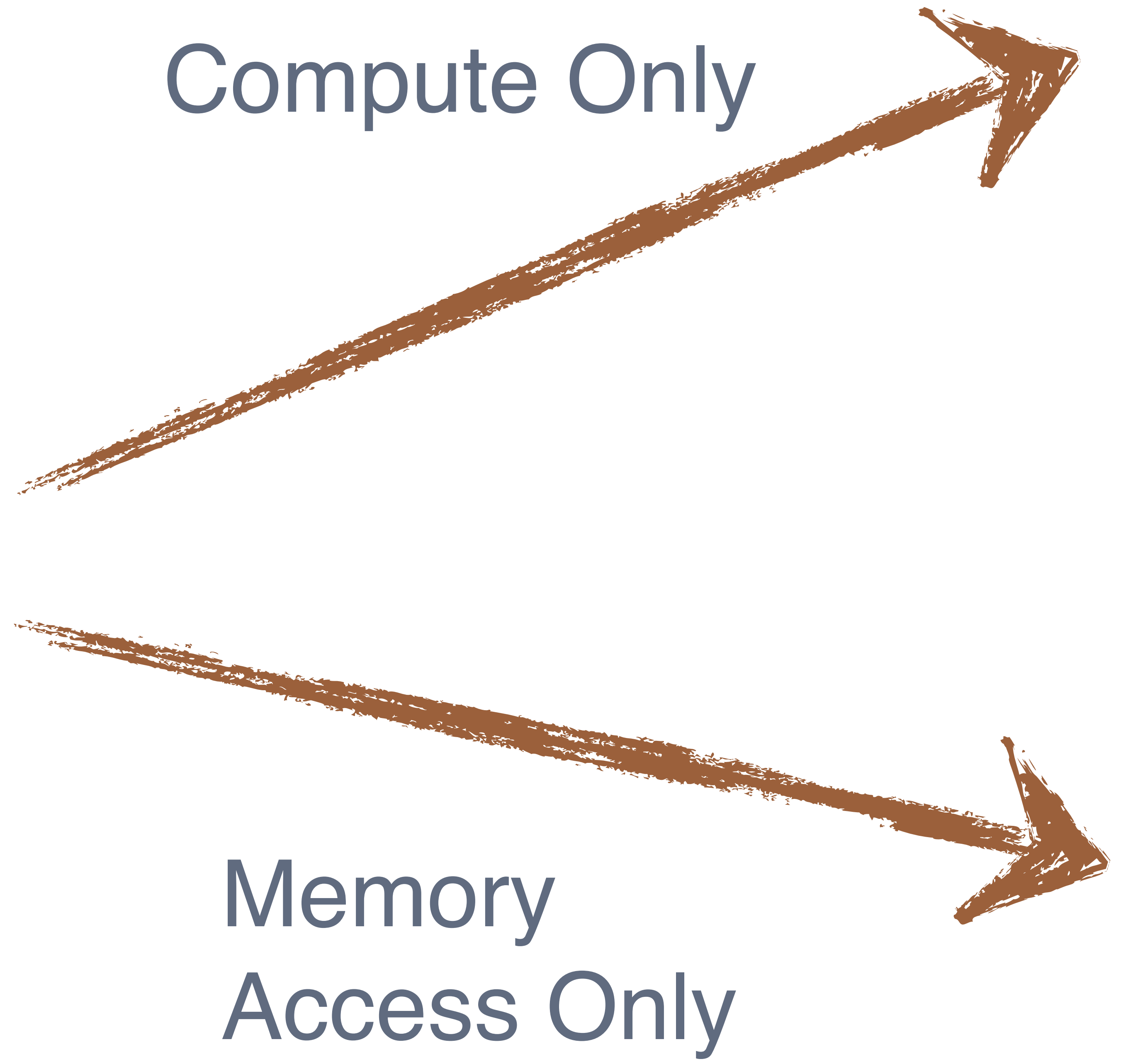
Rewrites need *not* necessarily be semantics preserving!

PPA: Conceptual Example

Tri-Diagonal Elimination

```
for ( i=1 ; i<n ; i++ ) {
     x[i] = z[i]*( y[i] - x[i-1] );
}
```

```
vmovsd    xmm1, [8+rsi+r12]
vmovsd    xmm2, [16+rsi+r12]
vsubsd    xmm0, xmm1, xmm0
vmulsd    xmm3, xmm0, [8+rsi+rbp]
vmovsd    [8+rsi+r13], xmm3
vsubsd    xmm4, xmm2, xmm3
vmulsd    xmm0, xmm4, [16+rsi+rbp]
vmovsd    [16+rsi+r13], xmm0
```



Compute Only

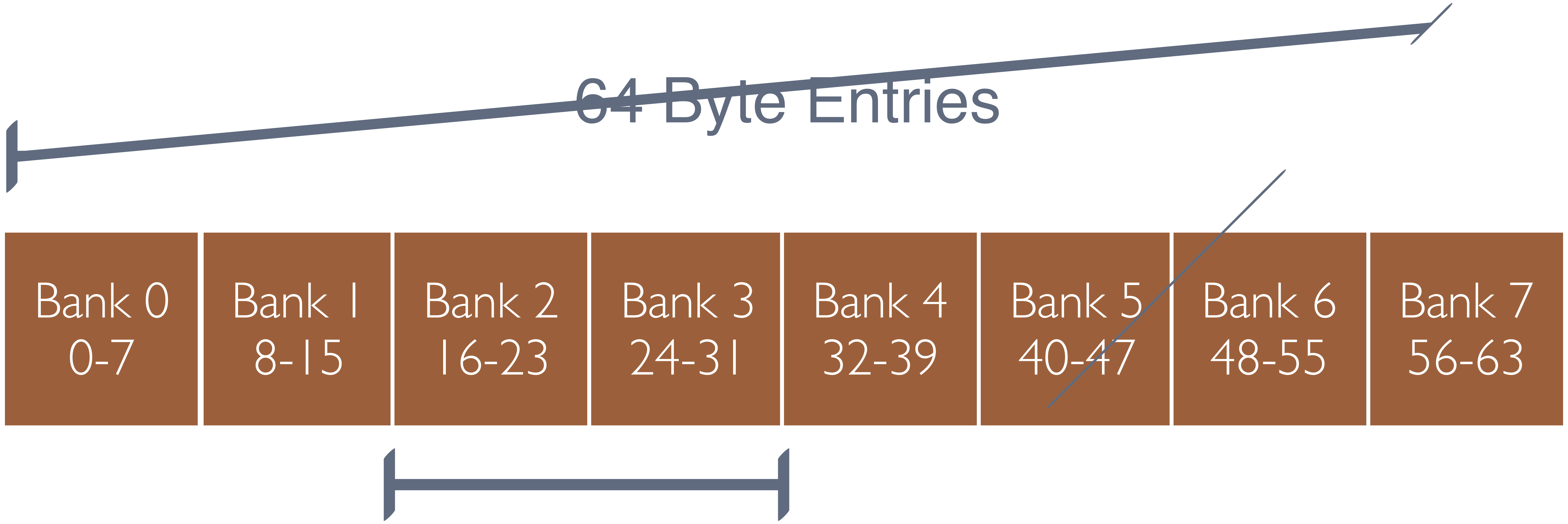
Memory Access Only

```
nop
nop
vsubsd    xmm0, xmm1, xmm0
vmulsd    xmm3, xmm0, xmm10
nop
vsubsd    xmm4, xmm2, xmm3
vmulsd    xmm0, xmm4, xmm12
nop
```

```
vmovsd    xmm1, [8+rsi+r12]
vmovsd    xmm2, [16+rsi+r12]
nop
vmovsd    xmm3, [8+rsi+rbp]
vmovsd    [8+rsi+r13], xmm3
nop
vmovsd    xmm0, [16+rsi+rbp]
vmovsd    [16+rsi+r13], xmm0
```

Perturbations do not need to preserve the semantic meaning

Concrete Example: L1D Bank Conflicts



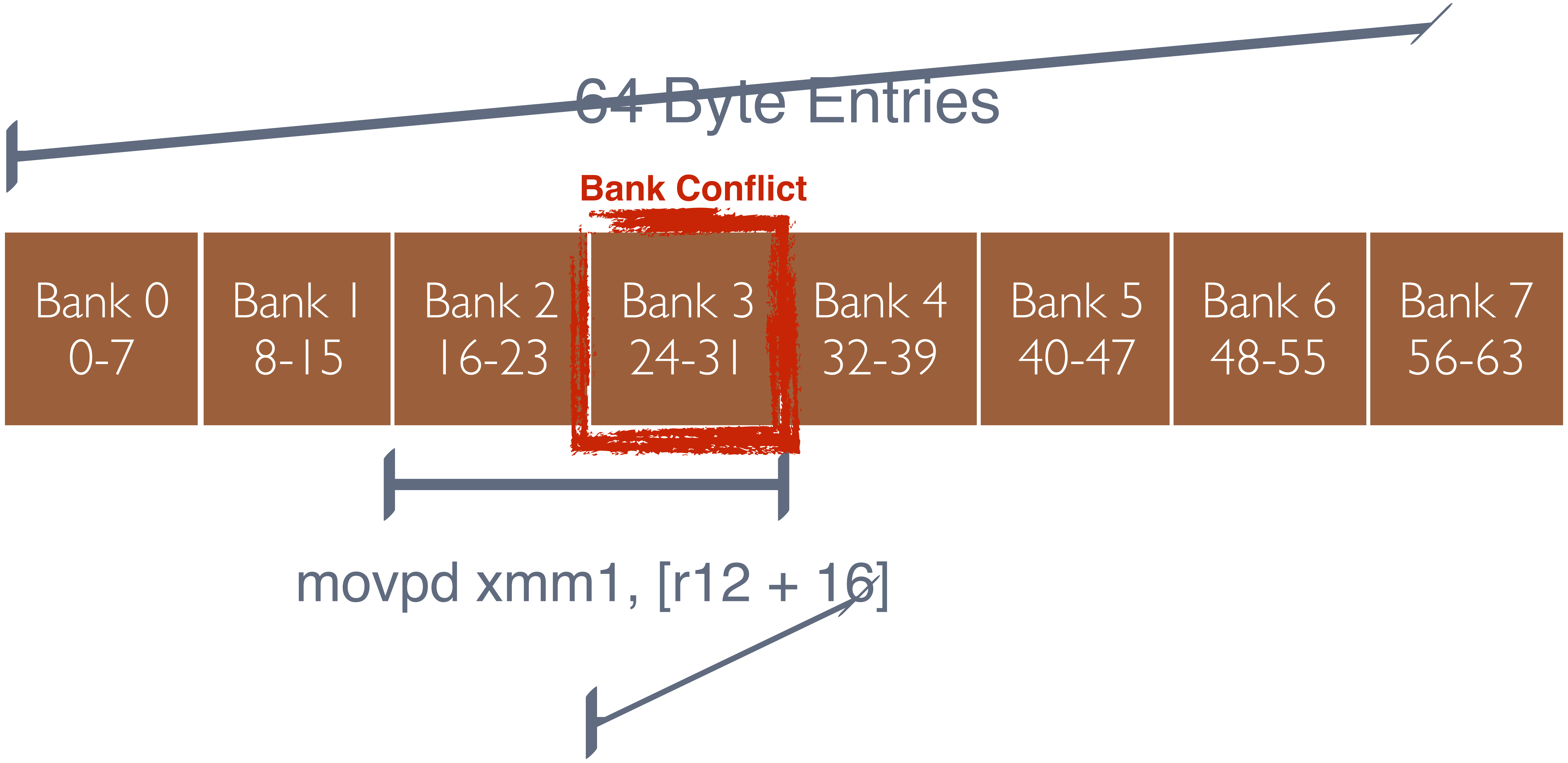
64 Byte Entries

Bank 0	Bank 1	Bank 2	Bank 3	Bank 4	Bank 5	Bank 6	Bank 7
0-7	8-15	16-23	24-31	32-39	40-47	48-55	56-63

movpd xmm1, [r12 + 16]

movpd xmm2, [r12 + 112]

CONCRETE EXAMPLE: L1D BANK CONFLICTS



64 Byte Entries

Bank Conflict

Bank 0 0-7	Bank 1 8-15	Bank 2 16-23	Bank 3 24-31	Bank 4 32-39	Bank 5 40-47	Bank 6 48-55	Bank 7 56-63

movpd xmm1, [r12 + 16]

movpd xmm2, [r12 + 88]

Concrete Example: L1D Bank Conflicts

```
[ 8 +rsi+r12]    ->    [X+rsi+r12]
[ 8 +rsi+r14]    ->    [X+rsi+r14]
[ 8 +rsi+rbp]    ->    [X+rsi+rbp]
[ 8 +rsi+r13]    ->    [X+rsi+r13]
[16+rsi+rbp]     ->    [X+rsi+rbp]
[16+rsi+r13]     ->    [X+rsi+r13]
```

*Assume rsi, r12, r13, r14, and rbp are 64-byte aligned

Original

```
vmovsd   xmm1, [8+rsi+r12]
vmovsd   xmm2, [8+rsi+r14]
vsubsd   xmm0, xmm1, xmm0
vmulsd   xmm3, xmm0, [8+rsi+rbp]
vmovsd   [8+rsi+r13], xmm3
vsubsd   xmm4, xmm2, xmm3
vmulsd   xmm0, xmm4, [16+rsi+rbp]
vmovsd   [16+rsi+r13], xmm0
```

Bank Conflicts ??

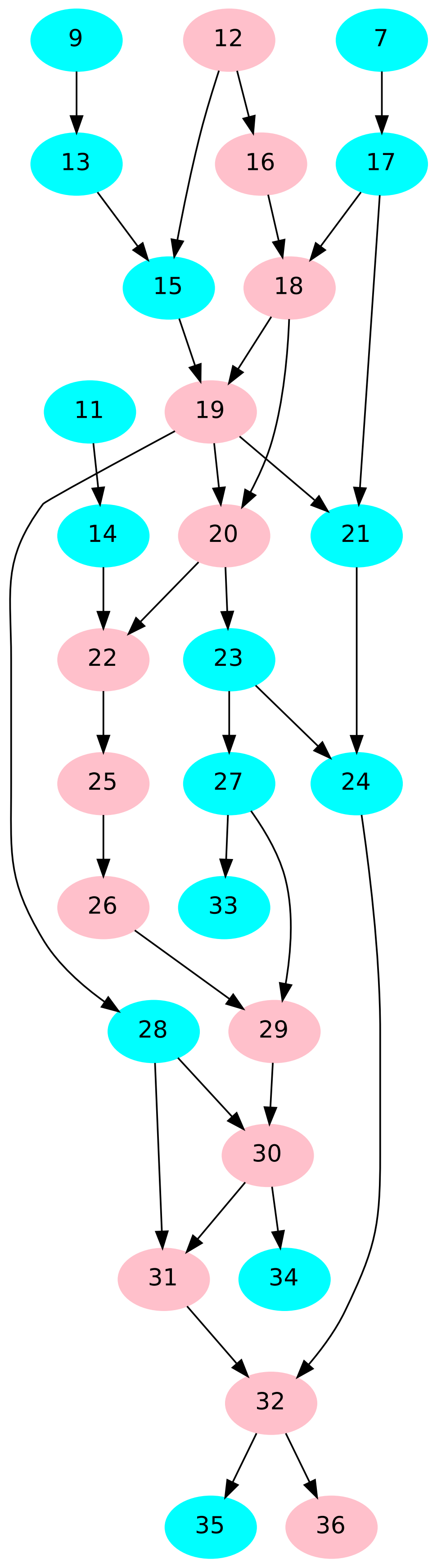
Perturbed Version

```
vmovsd   xmm1, [8+rsi+r12]
vmovsd   xmm2, [16+rsi+r14]
vsubsd   xmm0, xmm1, xmm0
vmulsd   xmm3, xmm0, [8+rsi+rbp]
vmovsd   [8+rsi+r13], xmm3
vsubsd   xmm4, xmm2, xmm3
vmulsd   xmm0, xmm4, [16+rsi+rbp]
vmovsd   [16+rsi+r13], xmm0
```

IDENTIFYING OOO-DEFICIENCIES

Original

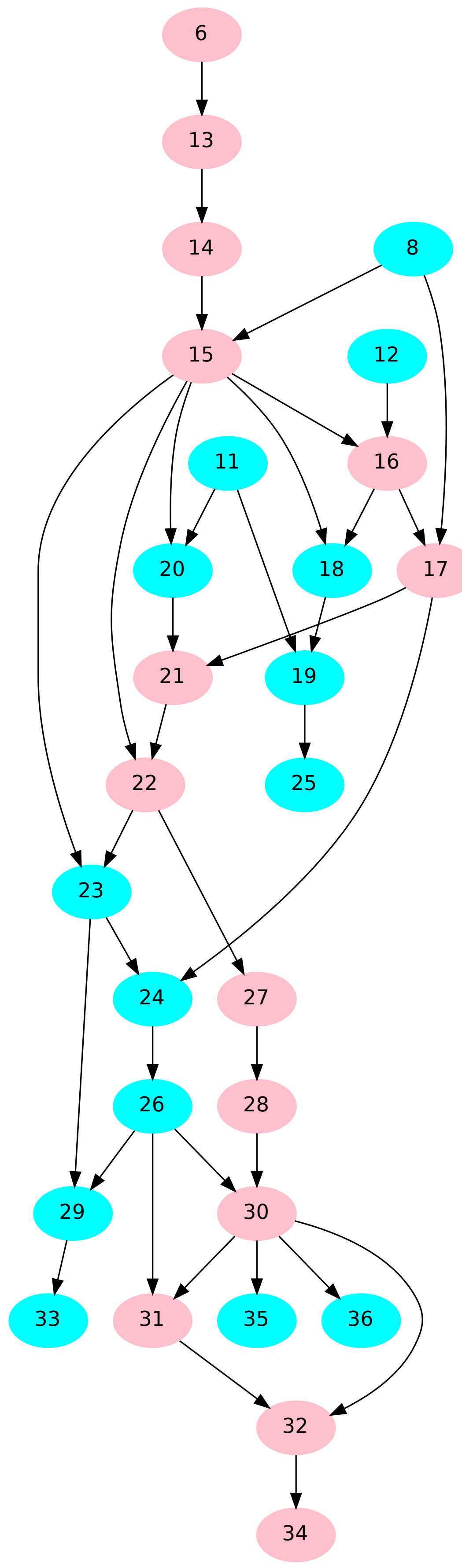
```
0  inloop:
1  movsd      xmm1, [88+r12+r9*8]
2  movsd      xmm1, [104+r12+r9*8]
3  movsd      xmm2, [120+r12+r9*8]
4  movsd      xmm2, [136+r12+r9*8]
5  movaps     xmm0, [80+r12+r9*8]
6  movhpd     xmm1, [96+r12+r9*8]
7  movaps     xmm2, [96+r12+r9*8]
8  movhpd     xmm3, [112+r12+r9*8]
9  movaps     xmm1, [112+r12+r9*8]
10 movhpd     xmm0, [128+r12+r9*8]
11 movaps     xmm0, [128+r12+r9*8]
12 movhpd     xmm3, [144+r12+r9*8]
13 mulpd      xmm1, xmm1
14 mulpd      xmm0, xmm0
15 mulpd      xmm1, xmm3
16 mulpd      xmm3, xmm3
17 mulpd      xmm2, xmm2
18 mulpd      xmm3, xmm2
19 mulpd      xmm1, xmm3
20 mulpd      xmm3, xmm1
21 addpd      xmm2, xmm1
22 addpd      xmm0, xmm3
23 addpd      xmm3, xmm3
24 addpd      xmm2, xmm3
25 mulpd      xmm0, [r15+r9*8]
26 mulpd      xmm0, [16+r15+r9*8]
27 mulpd      xmm3, [32+r15+r9*8]
28 mulpd      xmm1, [48+r15+r9*8]
29 addpd      xmm0, xmm3
30 addpd      xmm0, xmm1
31 addpd      xmm1, xmm0
32 addpd      xmm1, xmm2
33 movaps     [r11+r9*8], xmm3
34 movaps     [16+r11+r9*8], xmm0
35 movaps     [32+r11+r9*8], xmm1
36 movaps     [48+r11+r9*8], xmm1
37 add        r8, 1
38 cmp        r8, rbx
39 jb         inloop
```



Cycles per Iteration: **31.51 cycles**

Scrambled

```
0  inloop:
1  movsd      xmm2, [88+r12+r9*8]
2  movsd      xmm0, [104+r12+r9*8]
3  movsd      xmm0, [120+r12+r9*8]
4  movsd      xmm3, [136+r12+r9*8]
5  movaps     xmm0, [80+r12+r9*8]
6  movhpd     xmm3, [96+r12+r9*8]
7  movaps     xmm0, [96+r12+r9*8]
8  movhpd     xmm2, [112+r12+r9*8]
9  movaps     xmm1, [112+r12+r9*8]
10 movhpd     xmm0, [128+r12+r9*8]
11 movaps     xmm1, [128+r12+r9*8]
12 movhpd     xmm0, [144+r12+r9*8]
13 mulpd      xmm3, xmm3
14 mulpd      xmm3, xmm3
15 mulpd      xmm3, xmm2
16 mulpd      xmm0, xmm3
17 mulpd      xmm2, xmm0
18 mulpd      xmm0, xmm3
19 mulpd      xmm0, xmm1
20 mulpd      xmm1, xmm3
21 addpd      xmm1, xmm2
22 addpd      xmm1, xmm3
23 addpd      xmm3, xmm1
24 addpd      xmm2, xmm3
25 mulpd      xmm0, [r15+r9*8]
26 mulpd      xmm2, [16+r15+r9*8]
27 mulpd      xmm1, [32+r15+r9*8]
28 mulpd      xmm1, [48+r15+r9*8]
29 addpd      xmm3, xmm2
30 addpd      xmm1, xmm2
31 addpd      xmm2, xmm1
32 addpd      xmm2, xmm1
33 movaps     [r11+r9*8], xmm3
34 movaps     [16+r11+r9*8], xmm2
35 movaps     [32+r11+r9*8], xmm1
36 movaps     [48+r11+r9*8], xmm1
37 add        r8, 1
39 cmp        r8, rbx
39 jb         inloop
```



Cycles per Iteration: **19.65 cycles**

Our Vision for Performance Analysis

Can we account for all lost cycles?

```
for ( k=0 ; k<n ; k++ ) {
    x[k] = u[k] + r*( z[k] + r*y[k] ) +
        t*( u[k+3] + r*( u[k+2] + r*u[k+1] ) +
        t*( u[k+6] + r*( u[k+5] + r*u[k+4] ) ) );
}
```

→

Automated battery of experiments

- Frontend bottlenecks
- Scheduling resource conflicts
- Data bypass delays
- Cache latency stalls
- Memory disambiguation conflicts
- Retirement bandwidth

CONCLUSION / SUMMARY

Major Contribution: Active Performance Analysis

Status: Proof of concept

Gaps:

- Comprehensive set of experiments
- Scale beyond the core
- Generalize to additional microarchitectures

Cross-Pollination:

- Software optimization
- Autotuning and super-optimizing compilers
- Hardware-software codesign