GraphChi: Large-Scale Graph Computation on Just a PC

Aapo Kyrölä, Guy Blelloch, Carlos Guestin

CJ QUINES

GraphChi: **Large-Scale Graph** Computation on Just a PC

Aapo Kyrölä, Guy Blelloch, Carlos Guestin

CJ QUINES

So are “large-scale graphs” in the room with us right now?

So are “large-scale graphs” in the room with us right now?

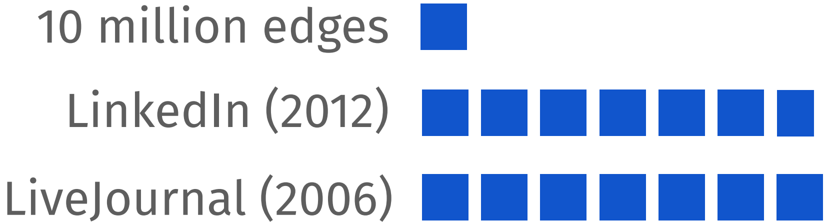
unfortunately…

big graphs are **real**

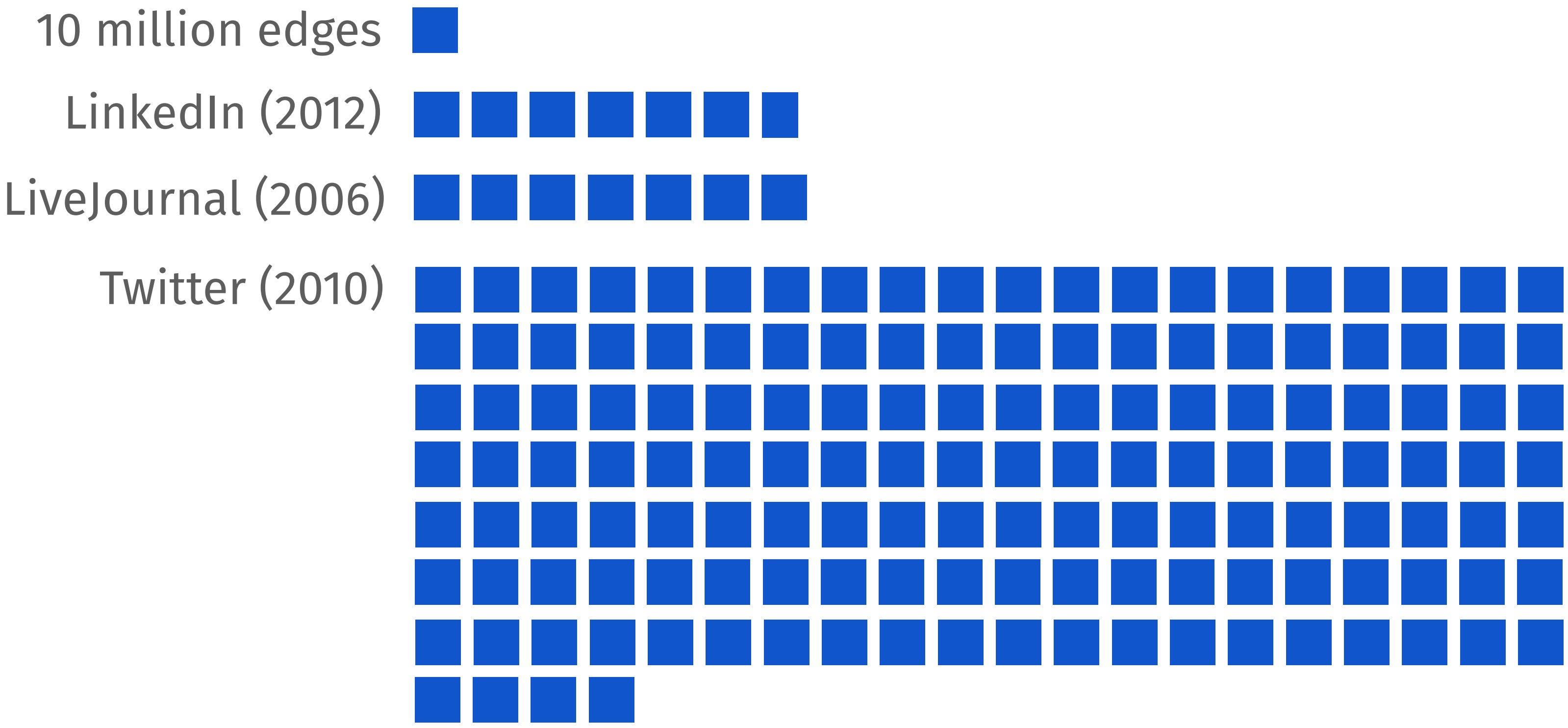
10 million edges 

big graphs are **real**

big graphs are **real**

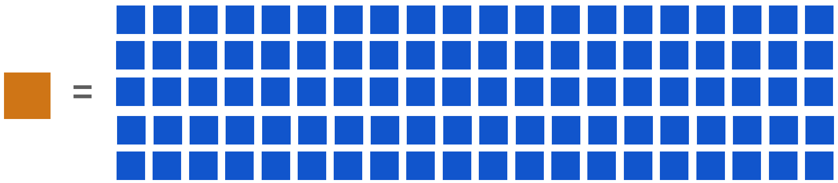


big graphs are **real**



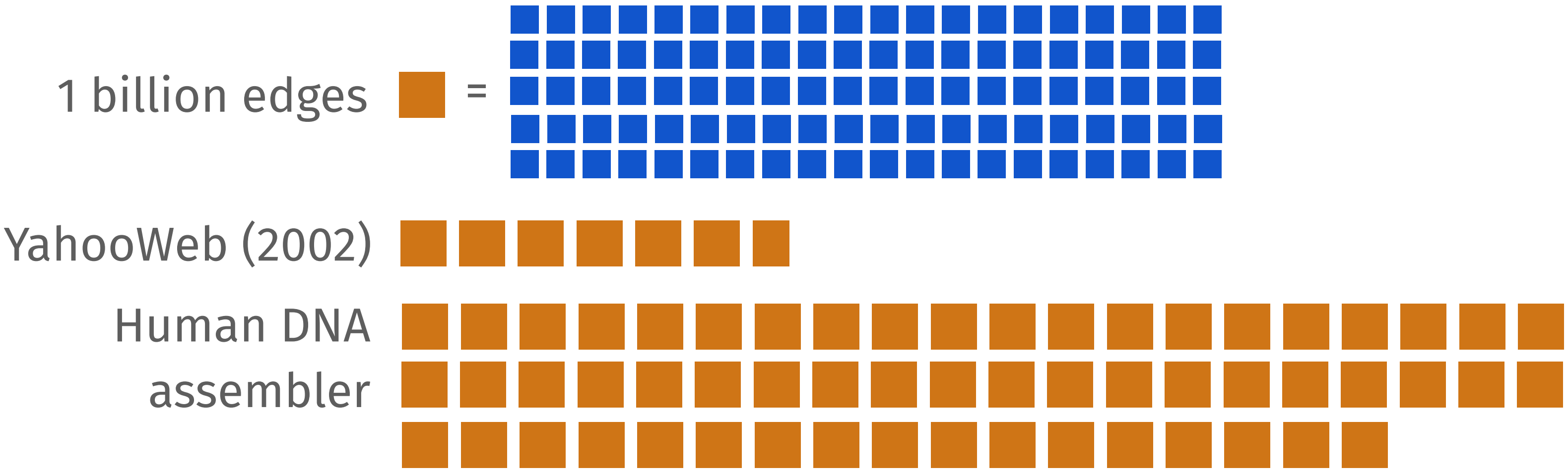
big graphs are **real**

1 billion edges

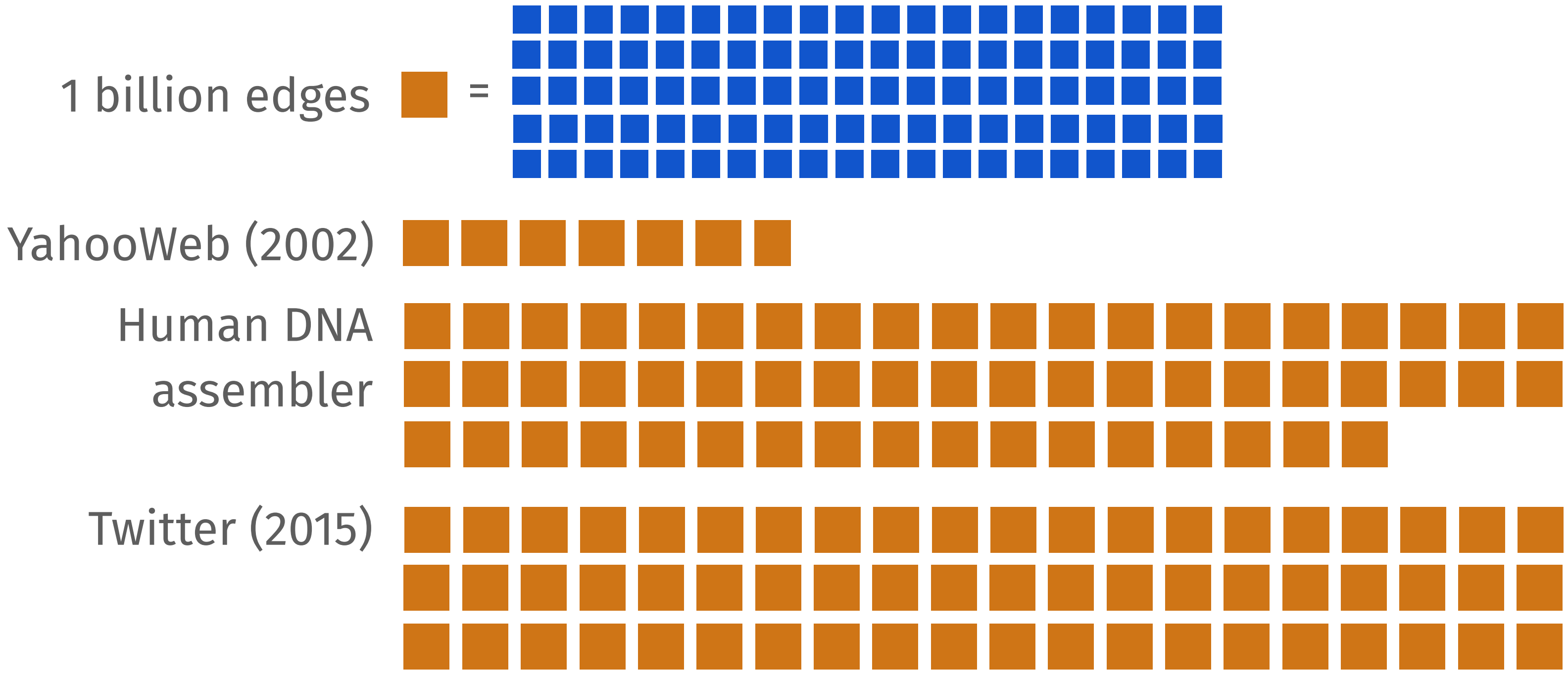


big graphs are **real**

big graphs are **real**

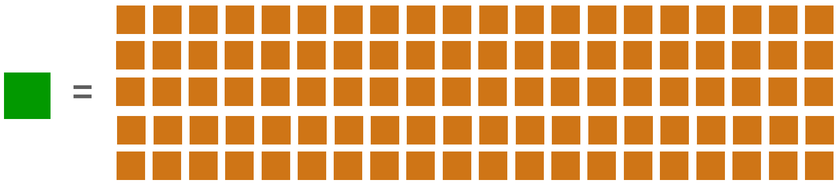


big graphs are **real**



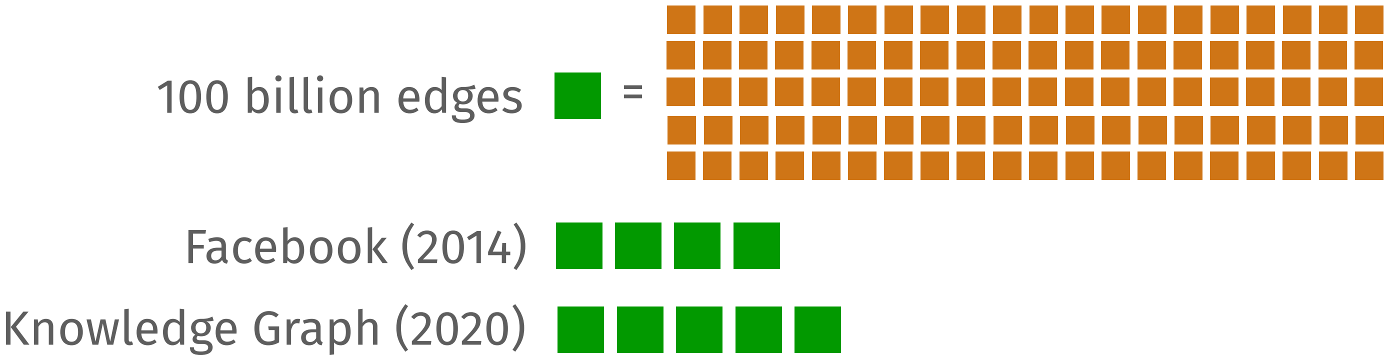
big graphs are **real**

100 billion edges

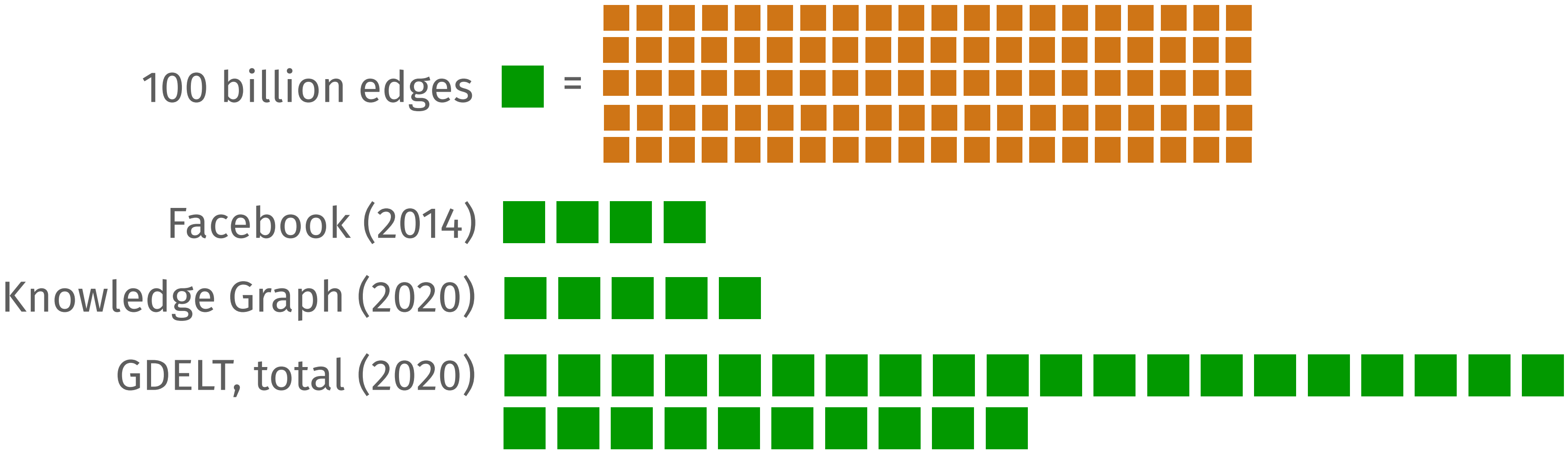


big graphs are **real**

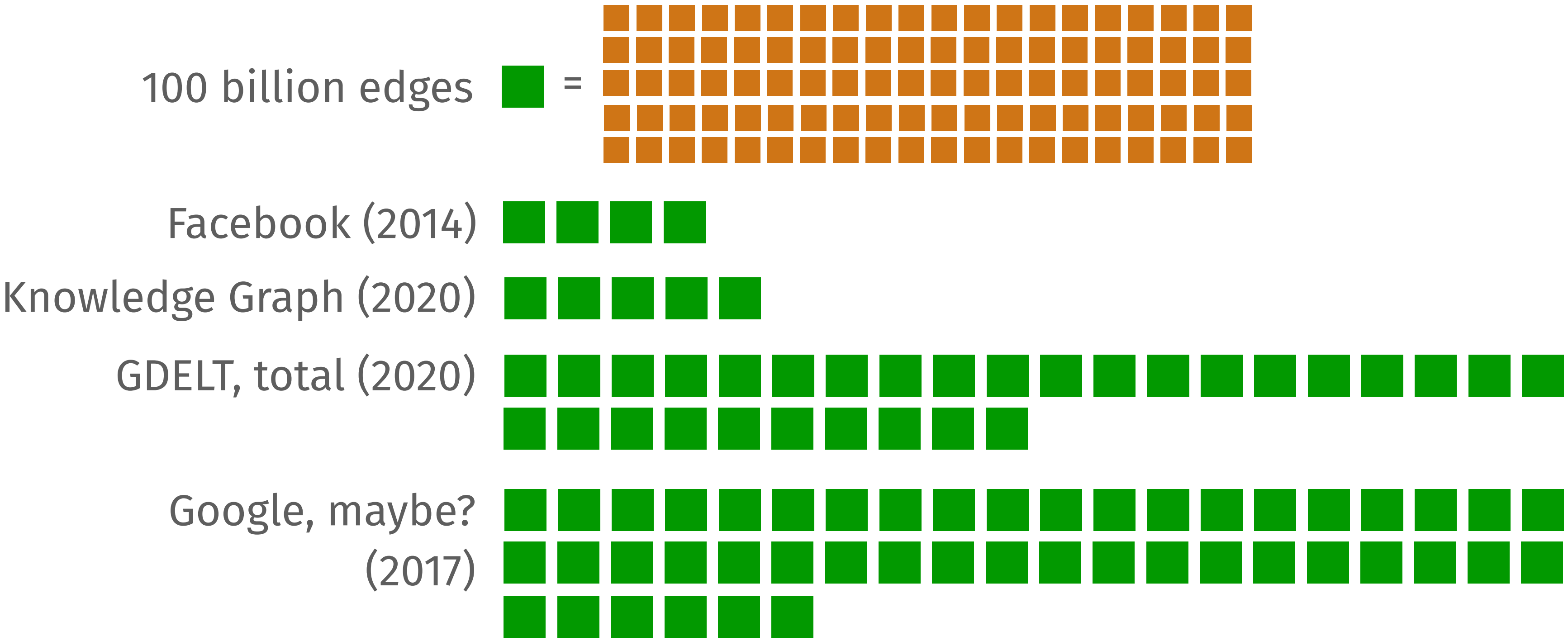
big graphs are **real**



big graphs are **real**



big graphs are **real**



GraphChi: **Large-Scale Graph** Computation on Just a PC

Aapo Kyrölä, Guy Blelloch, Carlos Guestin

CJ QUINES

GraphChi: Large-Scale Graph Computation on **Just a PC**

Aapo Kyrölä, Guy Blelloch, Carlos Guestin

CJ QUINES

who would process ■■■■■■■ first?

■ 10 million edges ■ 1 billion edges ■ 100 billion edges

who would process ■■■■■■■ first?

100 big computers

running Hadoop

10 million edges

1 billion edges

100 billion edges

who would process first?

100 big computers

running Hadoop

one small boi

running GraphChi

10 million edges

1 billion edges

100 billion edges

who would process ■■■■■■■ first?

100 big computers

running Hadoop

22 minutes

one small boi

running GraphChi

10 million edges   1 billion edges   100 billion edges

who would process ■■■■■■■ first?

100 big computers

running Hadoop

22 minutes

one small boi

running GraphChi

27 minutes

■ 10 million edges ■ 1 billion edges ■ 100 billion edges

who would process ■■■■■■■ first?

100 big computers

running Hadoop

22 minutes

one small boi

running GraphChi

27 minutes

10 million edges

1 billion edges

100 billion edges


i'm faster......
but at what cost?

you're faster, but at what cost?

Application & Graph	**Iter.**	**Comparative result**	**GraphChi (Mac Mini)**
Pagerank &	3	GraphLab[30] on AMD server (8 CPUs) **87 s**	**132 s**
Pagerank &	5	Spark [45] with 50 nodes (100 CPUs): **486.6 s**	**790 s**
Pagerank &	100	Stanford GPS, 30 EC2 nodes (60 virt. cores), **144 min**	approx. **581 min**
Pagerank &	1	Piccolo, 100 EC2 instances (200 cores) **70 s**	approx. **26 min**
Webgraph-BP &	1	Pegasus (Hadoop) on 100 machines: **22 min**	**27 min**
ALS &	10	GraphLab on AMD server: **4.7 min**	**9.8 min** (in-mem) **40 min** (edge-repl.)
Triangle-count &	-	Hadoop, 1636 nodes: **423 min**	**60 min**
Pagerank &	1	PowerGraph, 64 x 8 cores: **3.6 s**	**158 s**
Triange-count &	-	PowerGraph, 64 x 8 cores: **1.5 min**	**60 min**

10 million edges 1 billion edges 100 billion edges

GraphChi: Large-Scale Graph Computation on **Just a PC**

Aapo Kyrölä, Guy Blelloch, Carlos Guestin

CJ QUINES

GraphChi: Large-Scale Graph Computation on Just a PC

Aapo Kyrölä, Guy Blelloch, Carlos Guestin

CJ QUINES

obligatory outline slide

10 million edges 1 billion edges 100 billion edges

obligatory outline slide

- size limitations
  - can fit in a Mac Mini?

10 million edges   1 billion edges   100 billion edges

obligatory outline slide

- size limitations
  - can  fit in a Mac Mini?
- access pattern speed
  - “put it on disk and call it a day” doesn’t work

10 million edges 1 billion edges 100 billion edges

obligatory outline slide

- size limitations
  - can fit in a Mac Mini?
- access pattern speed
  - “put it on disk and call it a day” doesn’t work
- parallel sliding window
  - you could’ve built GraphChi (maybe)

10 million edges 1 billion edges 100 billion edges

size limitations

access pattern speed

parallel sliding window

can  fit in a Mac Mini?

10 million edges

1 billion edges

100 billion edges

can ■■■■■■■ fit in a Mac Mini?

- how big is an edge?

■ 10 million edges ■ 1 billion edges ■ 100 billion edges

can ■■■■■■■ fit in a Mac Mini?

- how big is an edge?
  - no compression; we're operating on edges in memory

■ 10 million edges ■ 1 billion edges ■ 100 billion edges

can ■■■■■■■ fit in a Mac Mini?

- how big is an edge?
  - no compression; we're operating on edges in memory
  - let's say it's 64 bits = 8 bytes

■ 10 million edges ■ 1 billion edges ■ 100 billion edges

can fit in a Mac Mini?

- how big is an edge?
  - no compression; we're operating on edges in memory
  - let's say it's 64 bits = 8 bytes
- that means = 10^7 edges = 80^7 bytes = 80 MB

10 million edges 1 billion edges 100 billion edges

can fit in a Mac Mini?

- how big is an edge?
  - no compression; we're operating on edges in memory
  - let's say it's 64 bits = 8 bytes
- that means = 10^7 edges = 80^7 bytes = 80 MB
- so = 100 = 8 GB

10 million edges   1 billion edges   100 billion edges

can ■■■■■■■ fit in a Mac Mini?

- how big is an edge?
  - no compression; we’re operating on edges in memory
  - let’s say it’s 64 bits = 8 bytes
- that means ■ = 10^7 edges = 80^7 bytes = 80 MB
- so ■ = 100 ■ = 8 GB
- and ■ = 100 ■ = 800 GB

■ 10 million edges ■ 1 billion edges ■ 100 billion edges

does our math work out?

10 million edges = 80 MB

1 billion edges = 8 GB

100 billion edges = 800 GB

does our math work out?

- the Mac Mini used in the paper has 8 GB of RAM

10 million edges = 80 MB

1 billion edges = 8 GB

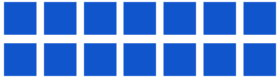
100 billion edges = 800 GB

does our math work out?

- the Mac Mini used in the paper has 8 GB of RAM
- according to our math, barely fits

10 million edges = 80 MB   1 billion edges = 8 GB   100 billion edges = 800 GB

does our math work out?

- the Mac Mini used in the paper has 8 GB of RAM
- according to our math,  barely fits
- according to the paper, it's around 

10 million edges = 80 MB   1 billion edges = 8 GB   100 billion edges = 800 GB

does our math work out?

- the Mac Mini used in the paper has 8 GB of RAM
- according to our math, barely fits
- according to the paper, it's around
- so we underestimated by an order of magnitude, why?

10 million edges = 80 MB 1 billion edges = 8 GB 100 billion edges = 800 GB

does our math work out?

- the Mac Mini used in the paper has 8 GB of RAM
- according to our math, barely fits
- according to the paper, it's around
- so we underestimated by an order of magnitude, why?
- answer: graphchi limits edges to 1 GB. why?

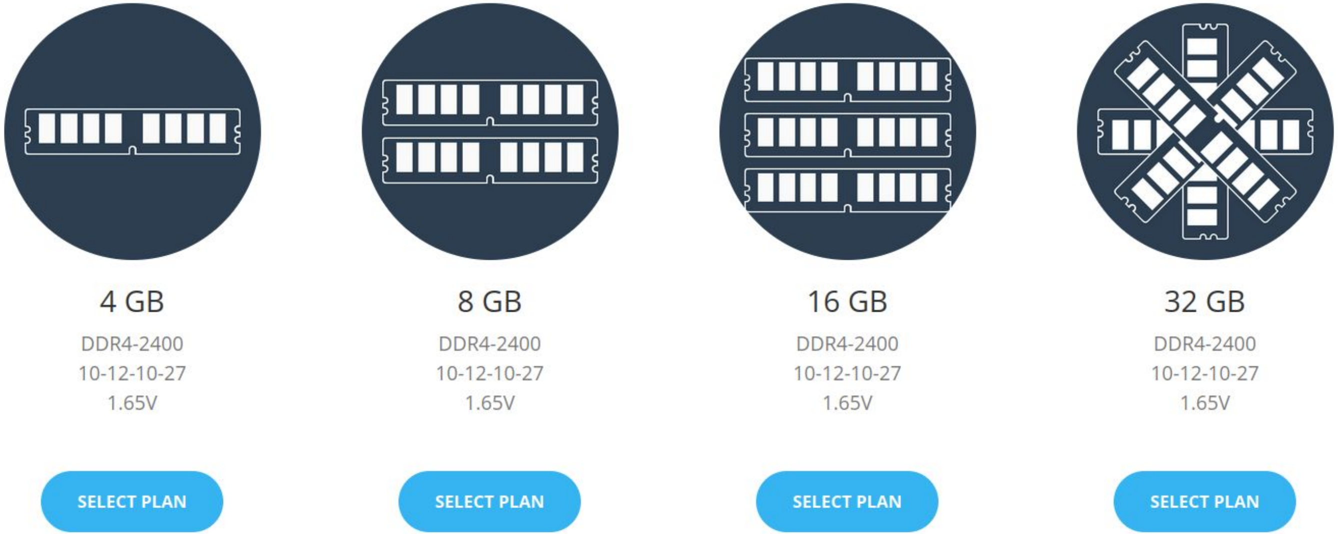
10 million edges = 80 MB   1 billion edges = 8 GB   100 billion edges = 800 GB

does our math work out?

- the Mac Mini used in the paper has 8 GB of RAM
- according to our math, barely fits
- according to the paper, it's around
- so we underestimated by an order of magnitude, why?
- answer: graphchi limits edges to 1 GB. why?
  - why won't more RAM help?

10 million edges = 80 MB   1 billion edges = 8 GB   100 billion edges = 800 GB

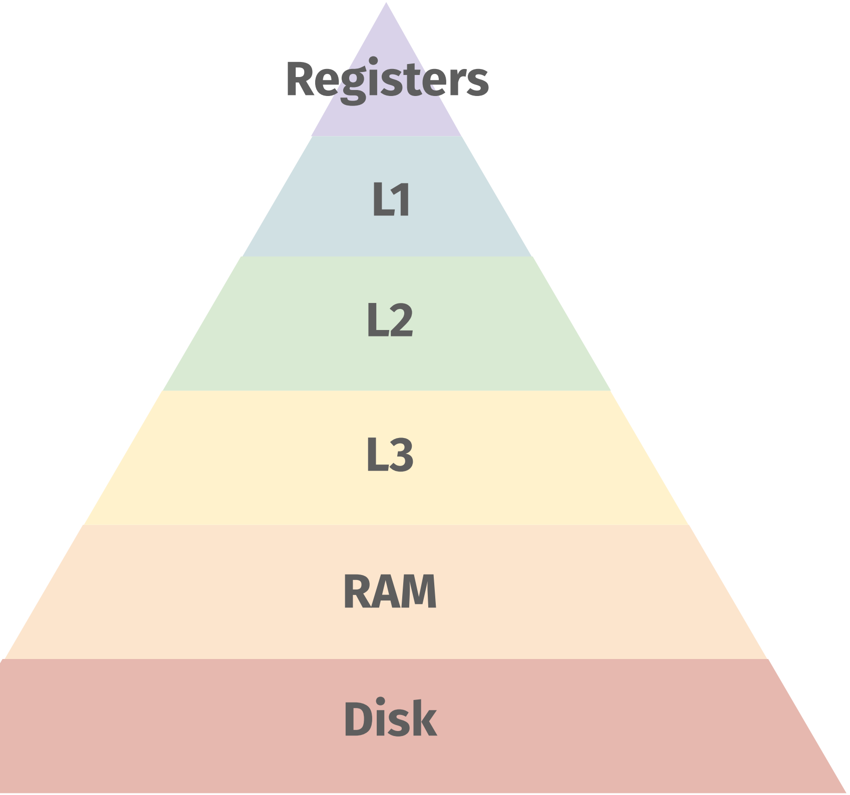
“just add more RAM”



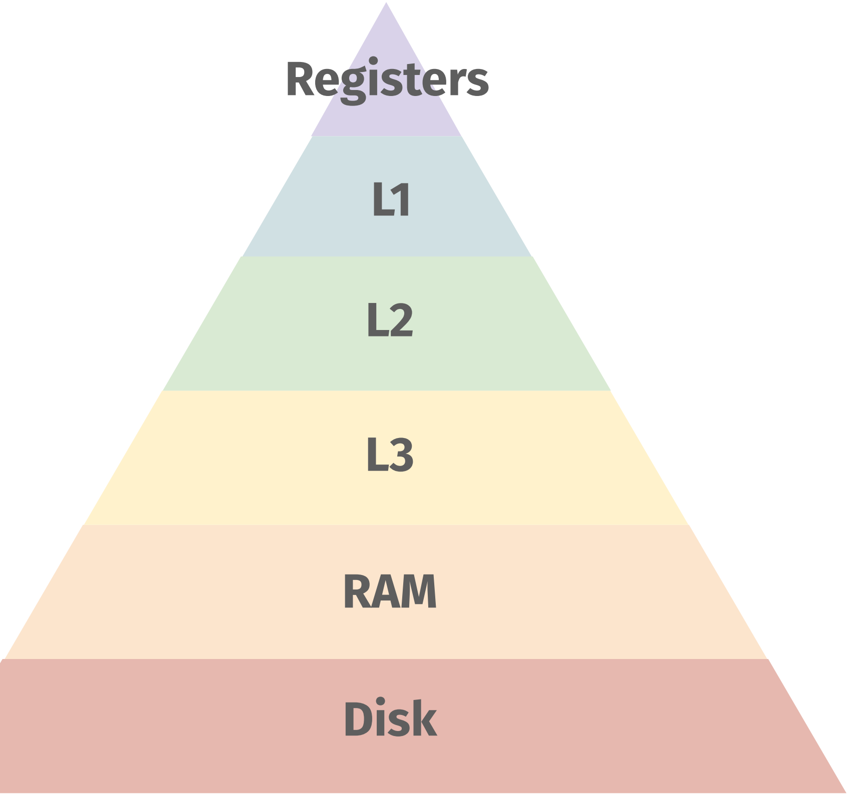
https://downloadmoreram.com

the memory hierarchy

the memory hierarchy



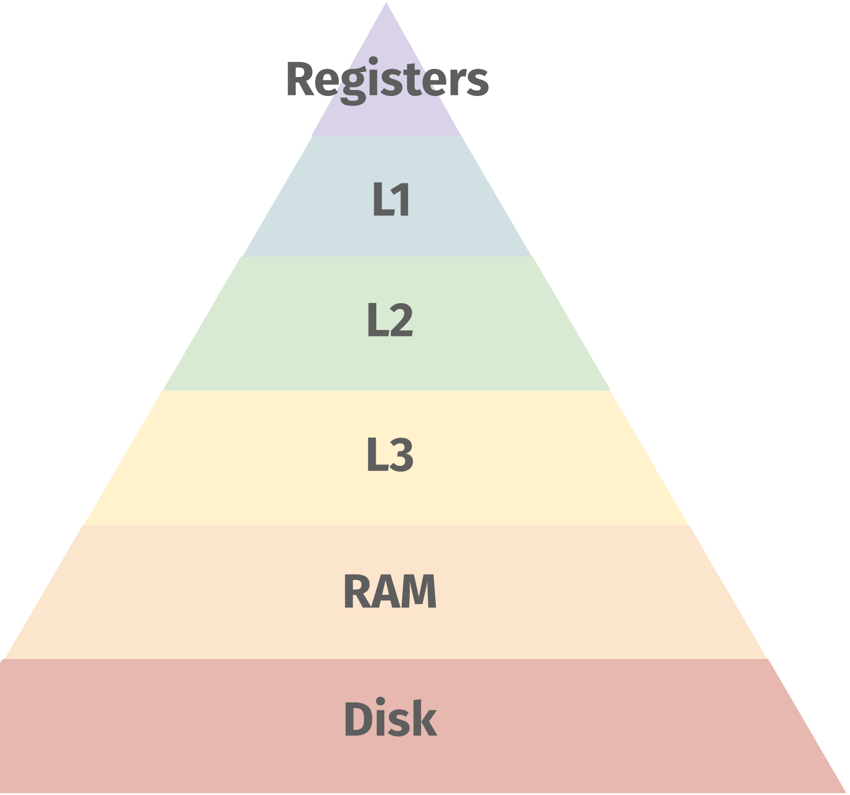
the memory hierarchy



latency
(1 ns = 10^{-9} s)

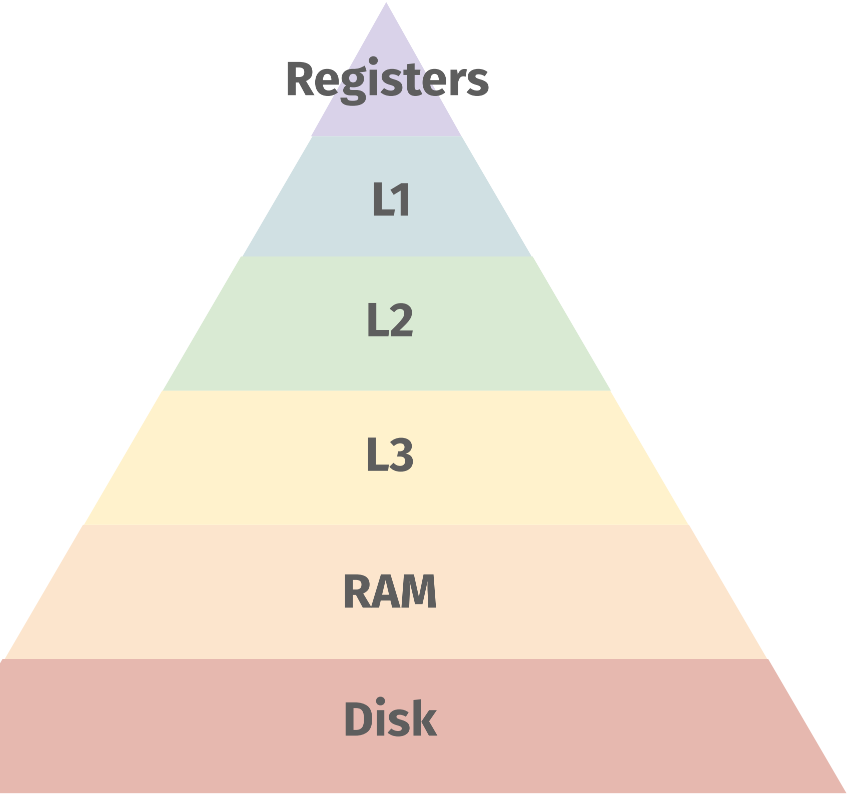
bandwidth
(GB/s = B/ns)

the memory hierarchy



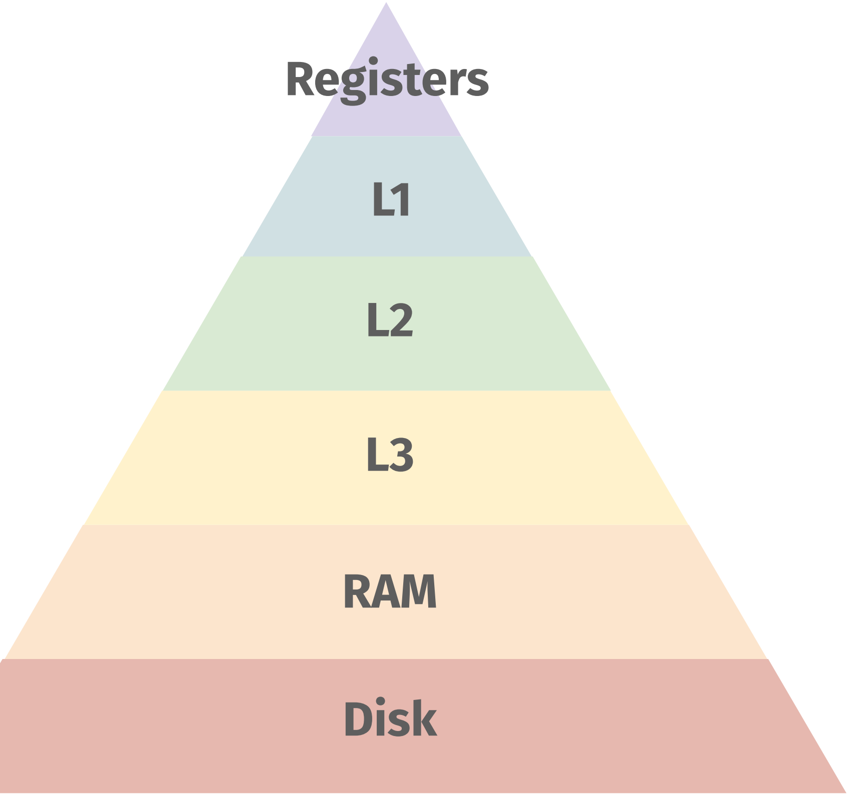
	latency (1 ns = 10^{-9} s)	bandwidth (GB/s = B/ns)
↔ L1	1 ns	25 GB/s
↔ L2	2.5 ns	15 GB/s
↔ L3	10 ns	10 GB/s

the memory hierarchy



	latency ($1\ \text{ns} = 10^{-9}\ \text{s}$)	bandwidth (GB/s = B/ns)
↔ L1	1 ns	25 GB/s
↔ L2	2.5 ns	15 GB/s
↔ L3	10 ns	10 GB/s
↔ RAM	50 ns	~~5 GB/s~~ (it's complicated)

the memory hierarchy



	latency (1 ns = 10^{-9} s)	bandwidth (GB/s = B/ns)
↔ L1	1 ns	25 GB/s
↔ L2	2.5 ns	15 GB/s
↔ L3	10 ns	10 GB/s
↔ RAM	50 ns	~~5 GB/s~~ (it's complicated)
↔ Disk	~~20,000 ns~~	~~0.1 GB/s~~
	(it's complicated)	

bandwidth is a limit too

10 million edges = 80 MB

1 billion edges = 8 GB

100 billion edges = 800 GB

bandwidth is a limit too

- more memory runs into diminishing returns

10 million edges = 80 MB   1 billion edges = 8 GB   100 billion edges = 800 GB

bandwidth is a limit too

- more memory runs into diminishing returns
- eventually, the bottleneck is bandwidth

10 million edges = 80 MB   1 billion edges = 8 GB   100 billion edges = 800 GB

bandwidth is a limit too

- more memory runs into diminishing returns
- eventually, the bottleneck is bandwidth
  - especially if we're compute-light:

10 million edges = 80 MB   1 billion edges = 8 GB   100 billion edges = 800 GB

bandwidth is a limit too

- more memory runs into diminishing returns
- eventually, the bottleneck is bandwidth
  - especially if we're compute-light:

Application	SSD	In-mem	Ratio
Connected components	45 s	18 s	2.5x
Community detection	110 s	46 s	2.4x
Matrix fact. (D=5, 5 iter)	114 s	65 s	1.8x
Matrix fact. (D=20, 5 iter.)	560 s	500 s	1.1x

10 million edges = 80 MB   1 billion edges = 8 GB   100 billion edges = 800 GB

bandwidth is a limit too

- more memory runs into diminishing returns
- eventually, the bottleneck is bandwidth
    - especially if we're compute-light:

Application	SSD	In-mem	Ratio
Connected components	45 s	18 s	2.5x
Community detection	110 s	46 s	2.4x
Matrix fact. (D=5, 5 iter)	114 s	65 s	1.8x
Matrix fact. (D=20, 5 iter.)	560 s	500 s	1.1x

    - so we can dump things on disk and call it a day right?

10 million edges = 80 MB   1 billion edges = 8 GB   100 billion edges = 800 GB

size limitations

access pattern speed

parallel sliding window

size limitations

access pattern speed

parallel sliding window

numbers i made up

- remember this?

numbers i made up

- remember this?

	latency (1 ns = 10^{-9} s)	bandwidth (GB/s = B/ns)
↔ RAM	50 ns	~~5 GB/s~~ (it's complicated)
↔ Disk	~~20,000 ns~~	~~0.1 GB/s~~

(it's complicated)

numbers i made up

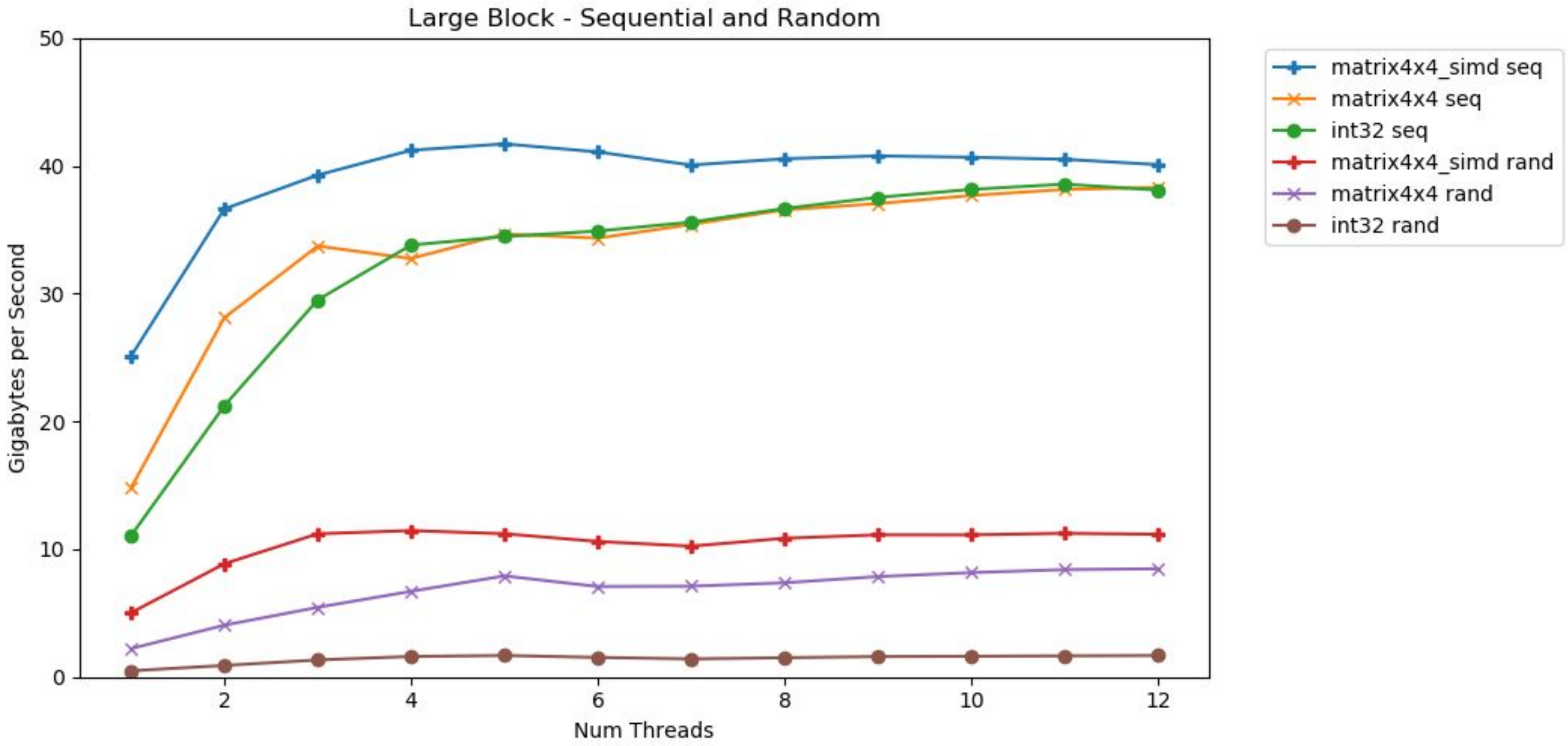
- remember this?

	latency (1 ns = 10^{-9} s)	bandwidth (GB/s = B/ns)
↔ RAM	50 ns	~~5 GB/s~~ (it's complicated)
↔ Disk	~~20,000 ns~~	~~0.1 GB/s~~

(it's complicated)

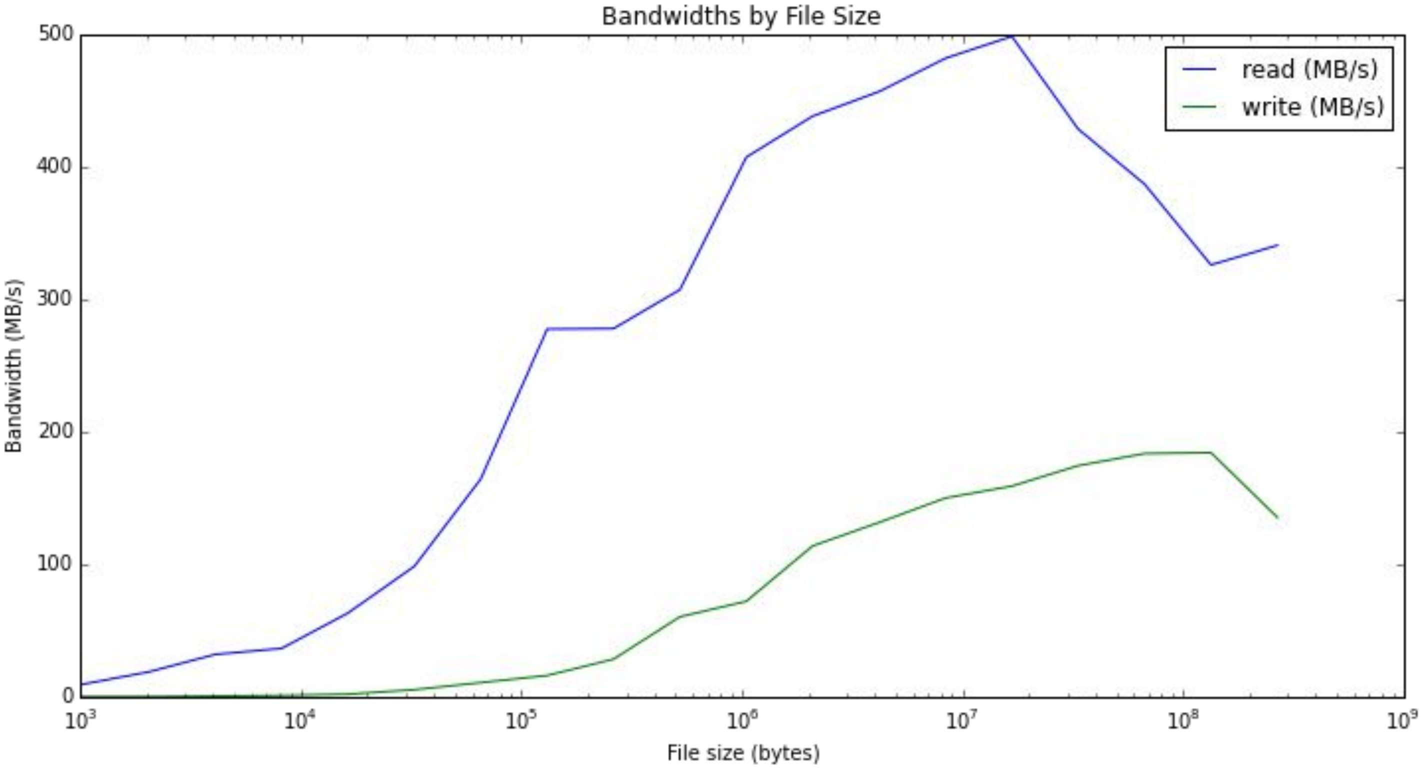
- it's complicated because of random vs. sequential access

RAM, read/write ints, look at ● and ●



SSD, read/write multiple files

random regime
(many small blocks)



sequential regime
(less large blocks)

numbers i mostly made up

numbers i mostly made up

	random		sequential	
	latency (1 ns = 10^{-9} s)	bandwidth (GB/s = B/ns)	latency (1 ns = 10^{-9} s)	bandwidth (GB/s = B/ns)
↔ RAM				
↔ SSD				
↔ HDD				

numbers i mostly made up

	random		sequential	
	latency (1 ns = 10^{-9} s)	bandwidth (GB/s = B/ns)	latency (1 ns = 10^{-9} s)	bandwidth (GB/s = B/ns)
↔ RAM	50 ns	0.5 GB/s		
↔ SSD	20,000 ns	0.005 GB/s		
↔ HDD	2,000,000 ns	0.001 GB/s		

numbers i mostly made up

	random		sequential	
	latency (1 ns = 10^{-9} s)	bandwidth (GB/s = B/ns)	latency (1 ns = 10^{-9} s)	bandwidth (GB/s = B/ns)
↔ RAM	50 ns	0.5 GB/s	50 ns	10 GB/s
↔ SSD	20,000 ns	0.005 GB/s	20,000 ns	0.5 GB/s
↔ HDD	2,000,000 ns	0.001 GB/s	500,000 ns	0.25 GB/s

random access problem

random access problem

- why can’t we “put it on disk and call it a day”?

random access problem

- why can’t we “put it on disk and call it a day”?
- the vertex-centric model has lots of random access patterns

analyzing memory access patterns

```
def update(v):
  for e in v.outedges:
    e.value = v.value
  v.value = sum(
    e.value for e in v.inedges
  )
```

analyzing memory access patterns

```
def update(v):
  for e in v.outedges:
    e.value = v.value
  v.value = sum(
    e.value for e in v.inedges
  )
```

? miss hit

analyzing memory access patterns

read

```
def update(v):
  for e in v.outedges:
    e.value = v.value
  v.value = sum(
    e.value for e in v.inedges
  )
```

? miss hit

analyzing memory access patterns

memory layout

read

```
def update(v):
  for e in v.outedges:
    e.value = v.value
  v.value = sum(
    e.value for e in v.inedges
  )
```

...
v
...

? miss hit

analyzing memory access patterns

memory layout

read

read

```
def update(v):
  for e in v.outedges:
    e.value = v.value
  v.value = sum(
    e.value for e in v.inedges
  )
```

…
v
…

? miss hit

analyzing memory access patterns

memory layout

read

read

```
def update(v):
  for e in v.outedges:
    e.value = v.value
  v.value = sum(
    e.value for e in v.inedges
  )
```

…

v.offset

…

…

e_1.dst

e_2.dst

…

CSR

? miss hit

analyzing memory access patterns

memory layout

read

read

read

```
def update(v):
  for e in v.outedges:
    e.value = v.value
  v.value = sum(
    e.value for e in v.inedges
  )
```

…
e₁.dst
e₂.dst
…

CSR

…
v.offset
…

? miss hit

analyzing memory access patterns

read

read

read

```
def update(v):
  for e in v.outedges:
    e.value = v.value
  v.value = sum(
    e.value for e in v.inedges
  )
```

memory layout

...
v.value
v.offset
...

...
e_1.dst
e_2.dst
...

CSR

? miss hit

analyzing memory access patterns

memory layout

read

read

read

```
def update(v):
  for e in v.outedges:
    e.value = v.value
  v.value = sum(
    e.value for e in v.inedges
  )
```

write
(looped)

…
v.value
v.offset
…

…
e_1.dst
e_2.dst
…

CSR

? miss hit

analyzing memory access patterns

memory layout

read

read

read

write (looped)

```
def update(v):
  for e in v.outedges:
    e.value = v.value
  v.value = sum(
    e.value for e in v.inedges
  )
```

…
v.value
v.offset
…

…
e_1.dst
e_1.value
e_2.dst
e_2.value
…

CSR

? miss hit

analyzing memory access patterns

read

read

```
def update(v):
  for e in v.outedges:
    e.value = v.value
  v.value = sum(
    e.value for e in v.inedges
  )
```

write (looped)

read

read

memory layout

…
v.value
v.offset
…

…
e_1.dst
e_1.value
e_2.dst
e_2.value
…

CSR

? miss hit

analyzing memory access patterns

read

read

read

write (looped)

```
def update(v):
  for e in v.outedges:
    e.value = v.value
  v.value = sum(
    e.value for e in v.inedges
  )
```

read

? miss hit

memory layout

…
v.value
$v.offset_1$
$v.offset_2$
…

…
e_1.dst
e_1.value
e_2.dst
e_2.value
…

CSR

…
f_1.src
f_1.value
f_2.src
f_2.value
…

CSC

analyzing memory access patterns

read

read

read

write (looped)

```
def update(v):
  for e in v.outedges:
    e.value = v.value
  v.value = sum(
    e.value for e in v.inedges
  )
```

read (looped)

read

memory layout

…
v.value
$v.offset_1$
$v.offset_2$
…

CSR

…
e_1.dst
e_1.value
e_2.dst
e_2.value
…

CSC

…
f_1.src
f_1.value
f_2.src
f_2.value
…

? miss hit

analyzing memory access patterns

read

read

read

write (looped)

```
def update(v):
  for e in v.outedges:
    e.value = v.value
  v.value = sum(
    e.value for e in v.inedges
  )
```

read (looped)

miss due to CSR/CSC sync

read

memory layout

…
v.value
$v.offset_1$
$v.offset_2$
…

…
e_1.dst
e_1.value
e_2.dst
e_2.value
…

CSR

synced values

CSC

…
f_1.src
f_1.value
f_2.src
f_2.value
…

? miss hit

analyzing memory access patterns

read

read

```
def update(v):
  for e in v.outedges:
    e.value = v.value
  v.value = sum(
    e.value for e in v.inedges
  )
```

read

write (looped)

write

read (looped)

miss due to CSR/CSC sync

read

memory layout

…
v.value
$v.offset_1$
$v.offset_2$
…

…
e_1.dst
e_1.value
e_2.dst
e_2.value
…

CSR

synced values

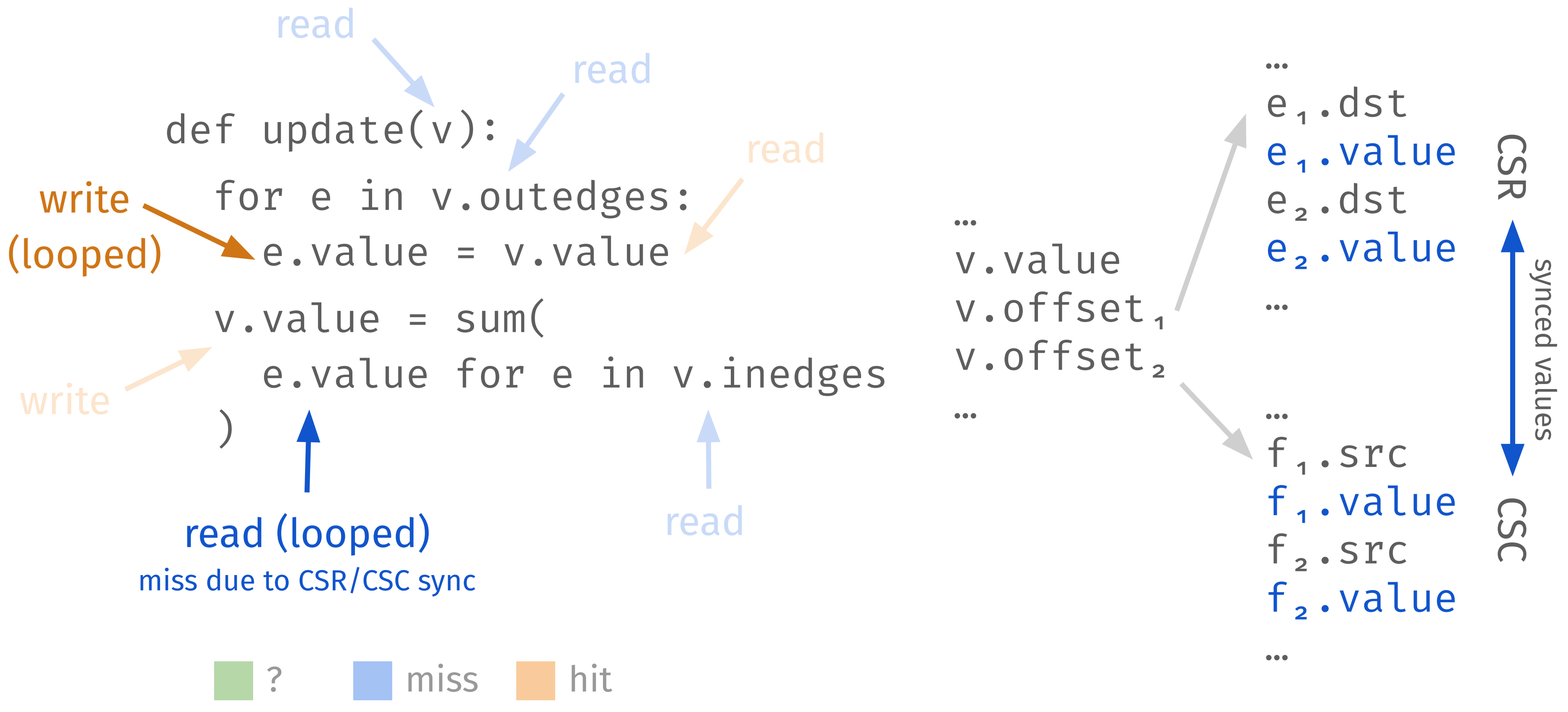
…
f_1.src
f_1.value
f_2.src
f_2.value
…

CSC

? miss hit

analyzing memory access patterns

memory layout



analyzing memory access patterns

read

read

```
def update(v):
  for e in v.outedges:
    e.value = v.value
  v.value = sum(
    e.value for e in v.inedges
  )
```

read

write (looped)

miss due to CSR/CSC sync

write

read (looped)

read

memory layout

...

v.value

$v.offset_1$

$v.offset_2$

...

...

e_1.dst

e_1.value

e_2.dst

e_2.value

...

CSR

synced values

CSC

...

f_1.src

f_1.value

f_2.src

f_2.value

...

? miss hit

random access problem

- why can’t we “put it on disk and call it a day”?
- the vertex-centric model has lots of random access patterns

random access problem

- why can't we "put it on disk and call it a day"?
- the vertex-centric model has lots of random access patterns
  - particularly, accessing out/inedges, as values need to be synced

random access problem

- why can’t we “put it on disk and call it a day”?
- the vertex-centric model has lots of random access patterns
    - particularly, accessing out/inedges, as values need to be synced
    - we can put outedges together, but no guarantee for inedges

random access problem

- why can't we "put it on disk and call it a day"?
- the vertex-centric model has lots of random access patterns
  - particularly, accessing out/inedges, as values need to be synced
  - we can put outedges together, but no guarantee for inedges
  - (or vice versa: inedges together, no guarantee for outedges)

random access problem

- why can’t we “put it on disk and call it a day”?
- the vertex-centric model has lots of random access patterns
  - particularly, accessing out/inedges, as values need to be synced
  - we can put outedges together, but no guarantee for inedges
  - (or vice versa: inedges together, no guarantee for outedges)
- the authors call this the random access problem

random access problem

- why can’t we “put it on disk and call it a day”?
- the vertex-centric model has lots of random access patterns
  - particularly, accessing out/inedges, as values need to be synced
  - we can put outedges together, but no guarantee for inedges
  - (or vice versa: inedges together, no guarantee for outedges)
- the authors call this the random access problem
  - not a problem for e.g. GraphLab, because everything’s in memory

random access problem

- why can’t we “put it on disk and call it a day”?
- the vertex-centric model has lots of random access patterns
  - particularly, accessing out/inedges, as values need to be synced
  - we can put outedges together, but no guarantee for inedges
  - (or vice versa: inedges together, no guarantee for outedges)
- the authors call this the random access problem
  - not a problem for e.g. GraphLab, because everything’s in memory
  - (recall: SSD random access is 100× slower than RAM random access)

size limitations

access pattern speed

parallel sliding window

ACT III

size limitations

access pattern speed

parallel sliding window

motivation

motivation

- assume we keep inedges sequentially

motivation

- assume we keep inedges sequentially
- we have to access outedges randomly

motivation

- assume we keep inedges sequentially
- we have to access outedges randomly
    - which is a problem if they’re all on disk

motivation

- assume we keep inedges sequentially
- we have to access outedges randomly
  - which is a problem if they're all on disk
- but random access isn't a problem if it's in memory!

motivation

- assume we keep inedges sequentially
- we have to access outedges randomly
    - which is a problem if they're all on disk
- but random access isn't a problem if it's in memory!
- so move all the outedges we need to memory first

motivation

- assume we keep inedges sequentially
- we have to access outedges randomly
  - which is a problem if they're all on disk
- but random access isn't a problem if it's in memory!
- so move all the outedges we need to memory first
  - it can't all fit, so do it one subgraph at a time

parallel sliding window

parallel sliding window

parallel sliding window

vertices 1–2

vertices 3–4

vertices 5–6

parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph

parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph

parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

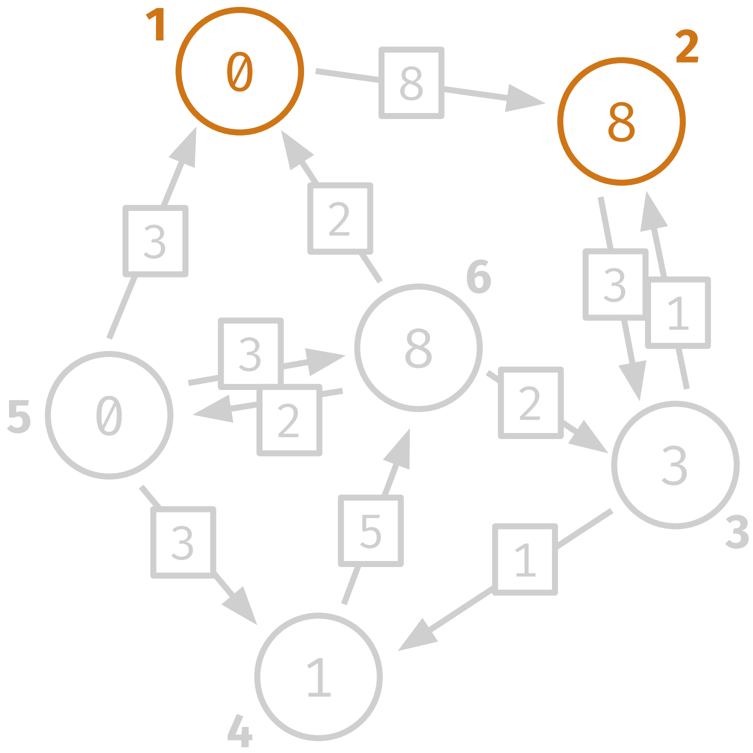
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

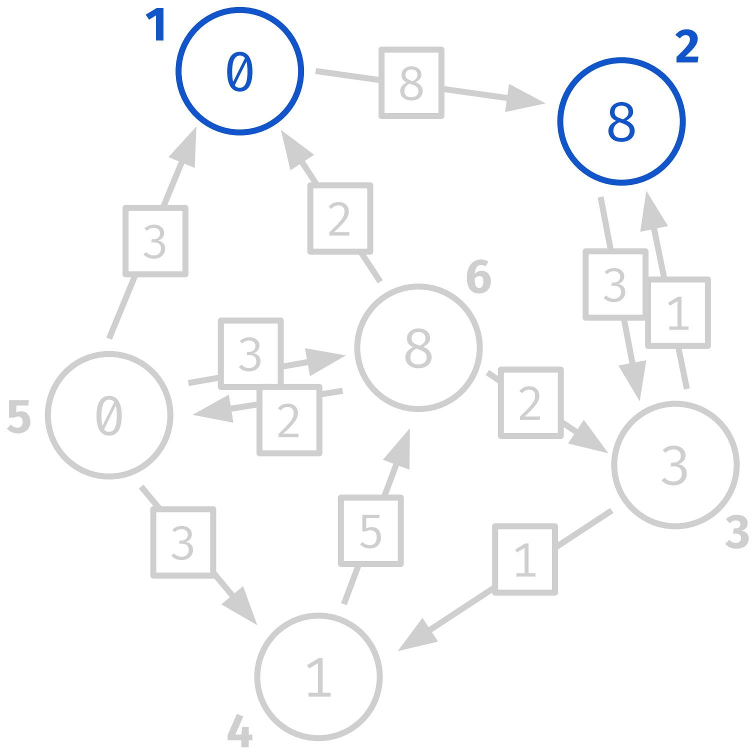
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

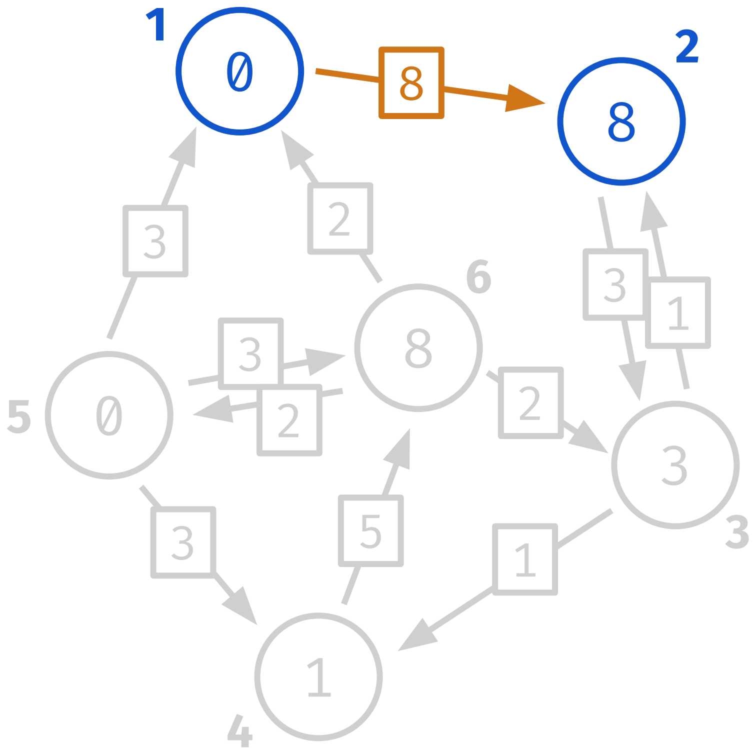
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

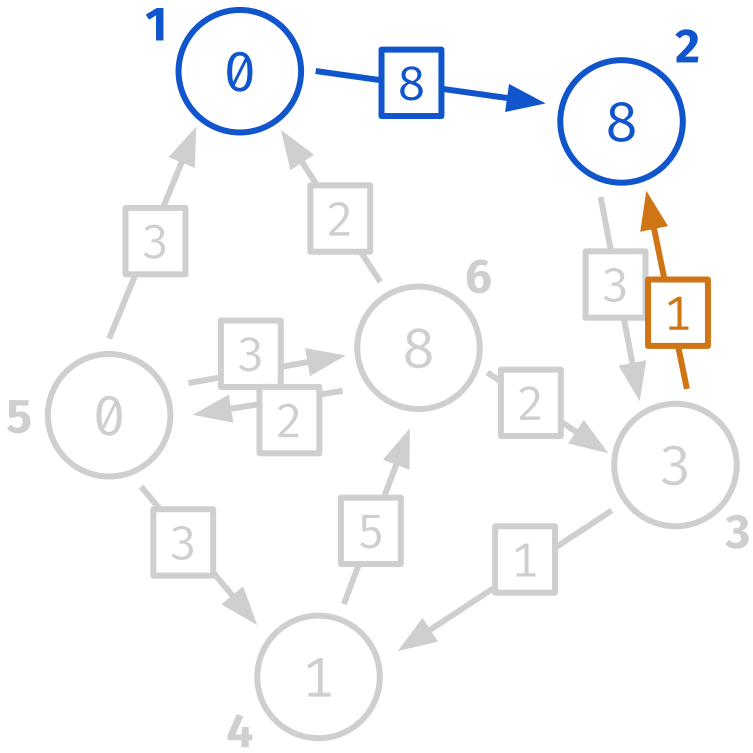
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

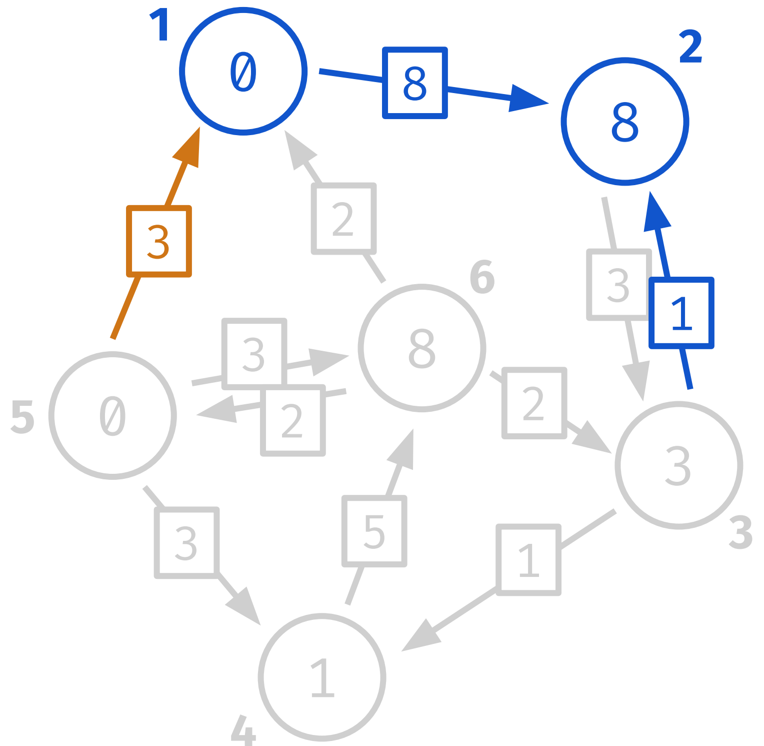
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

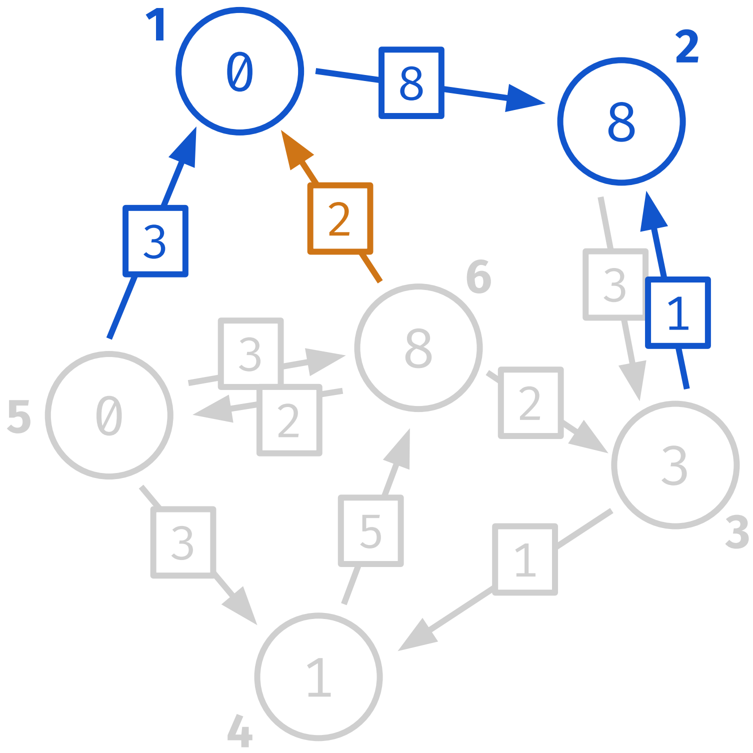
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

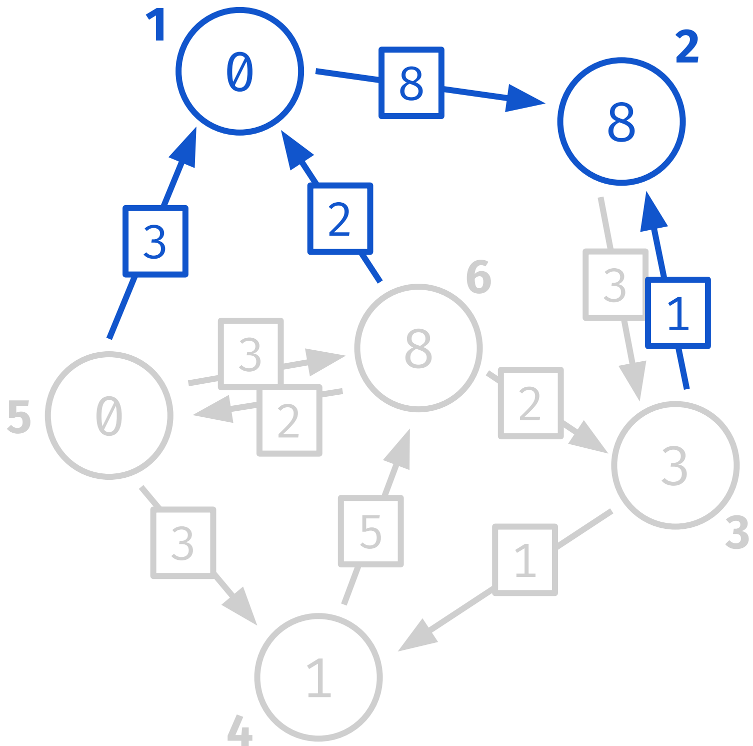
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

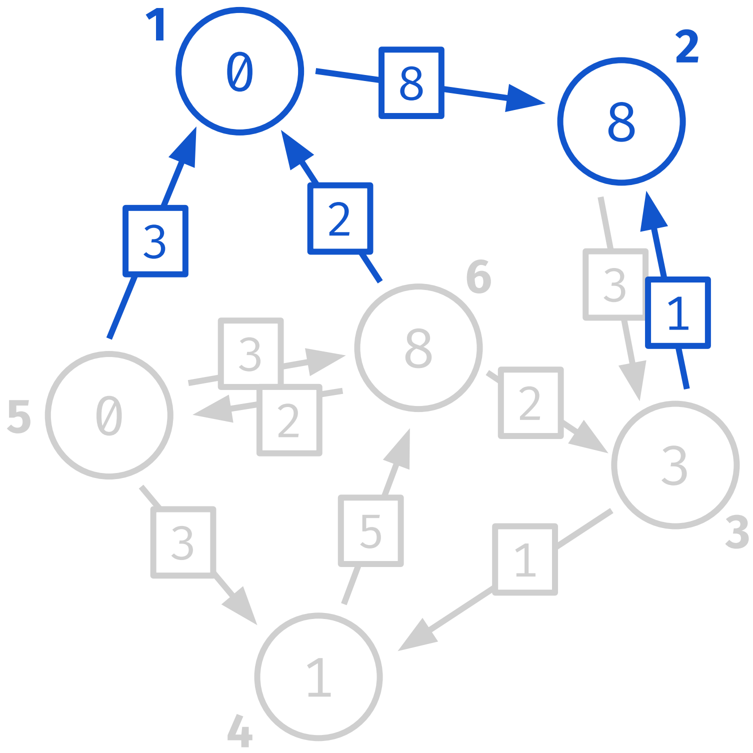
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

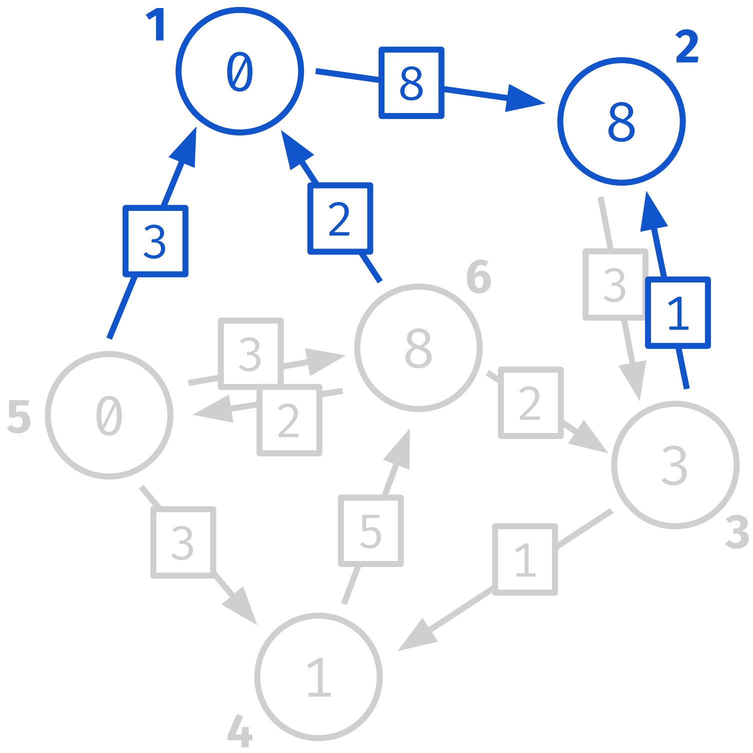
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

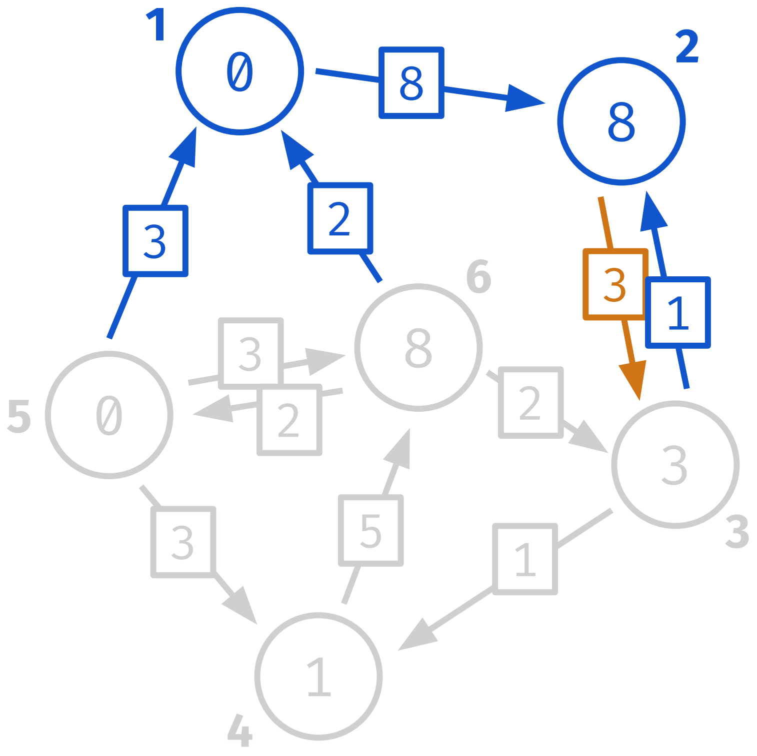
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

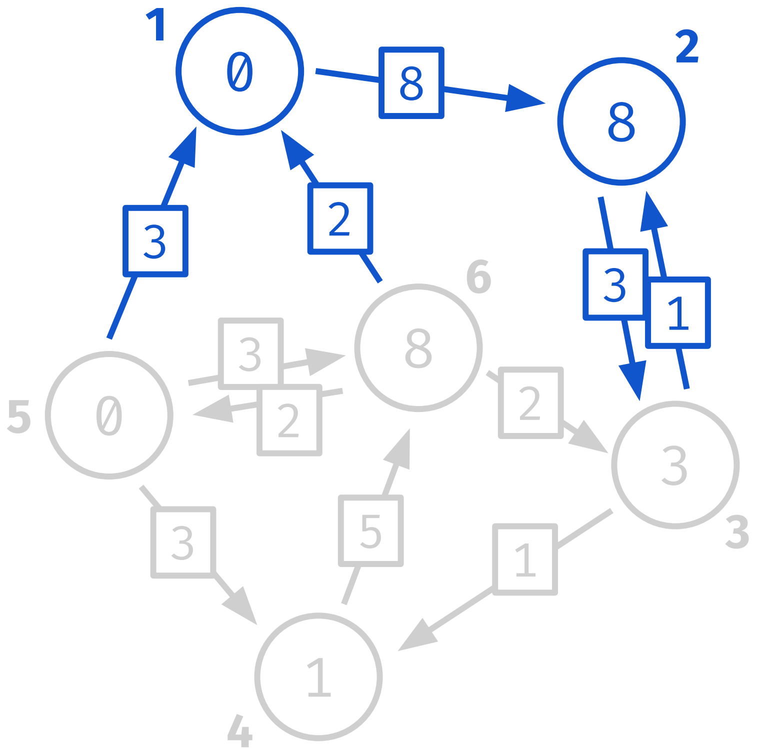
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph

parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

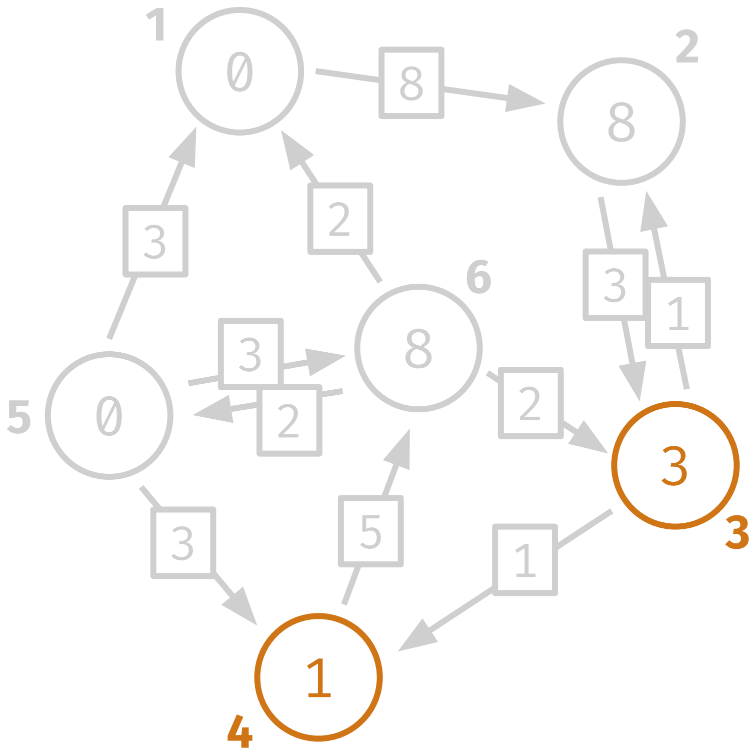
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

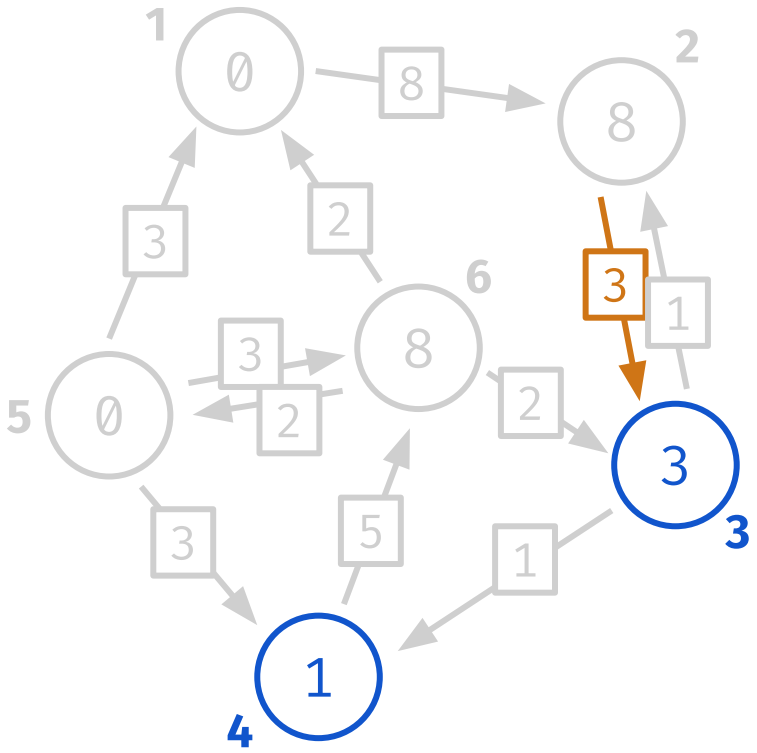
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

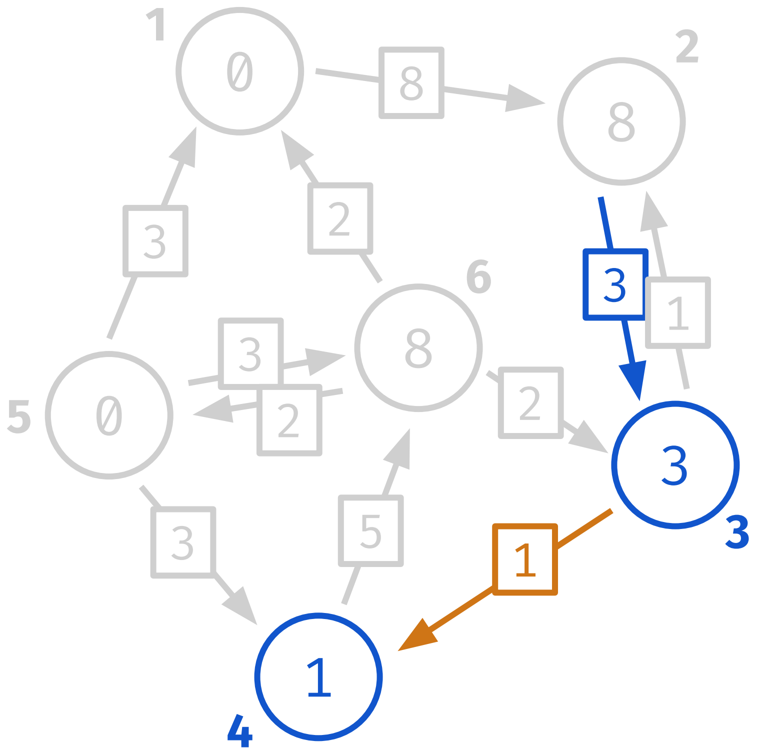
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

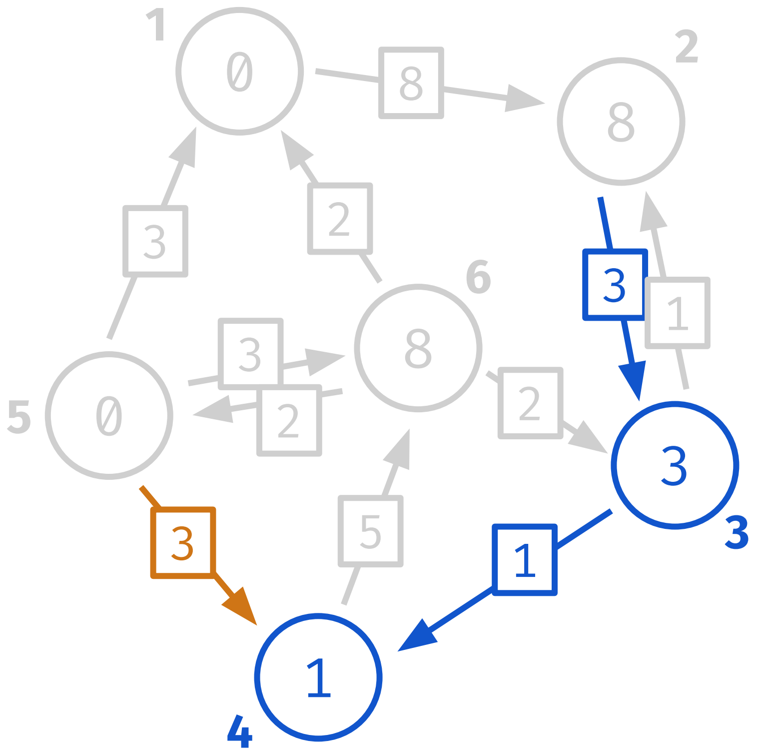
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

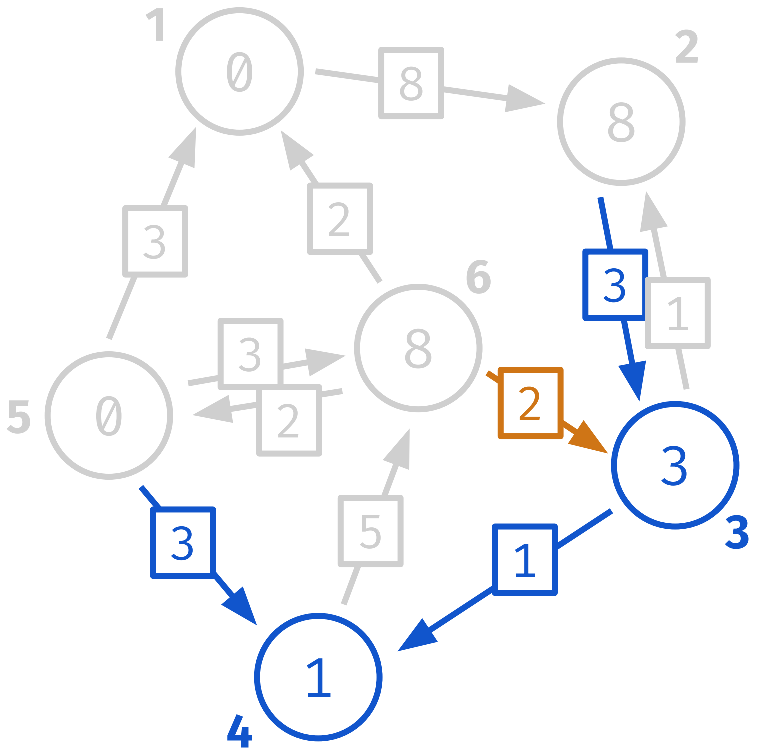
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

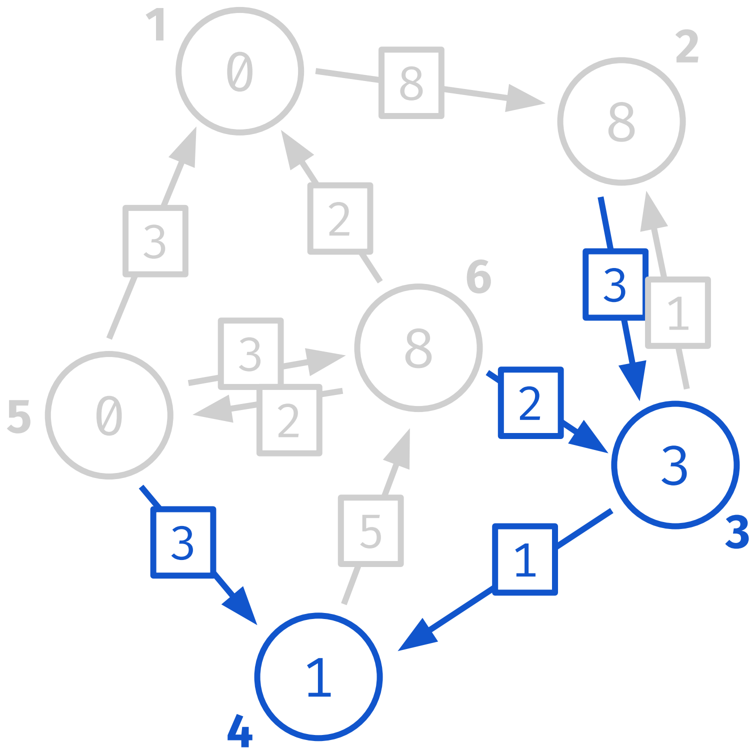
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

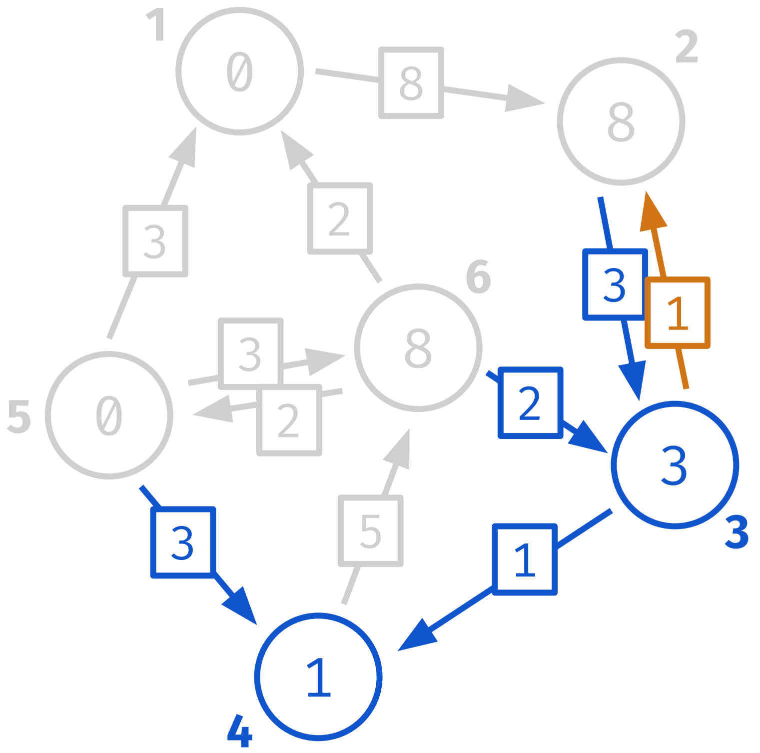
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

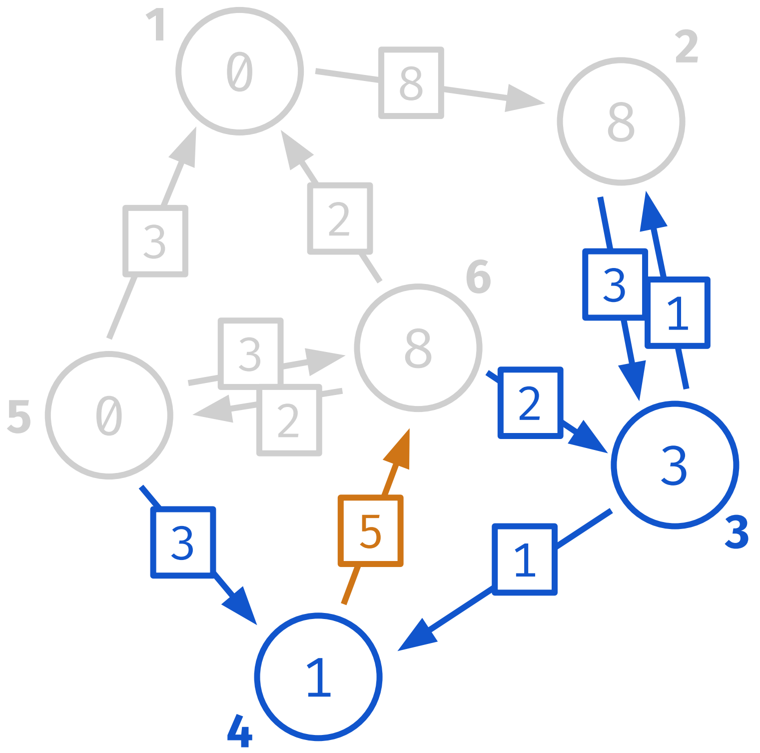
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

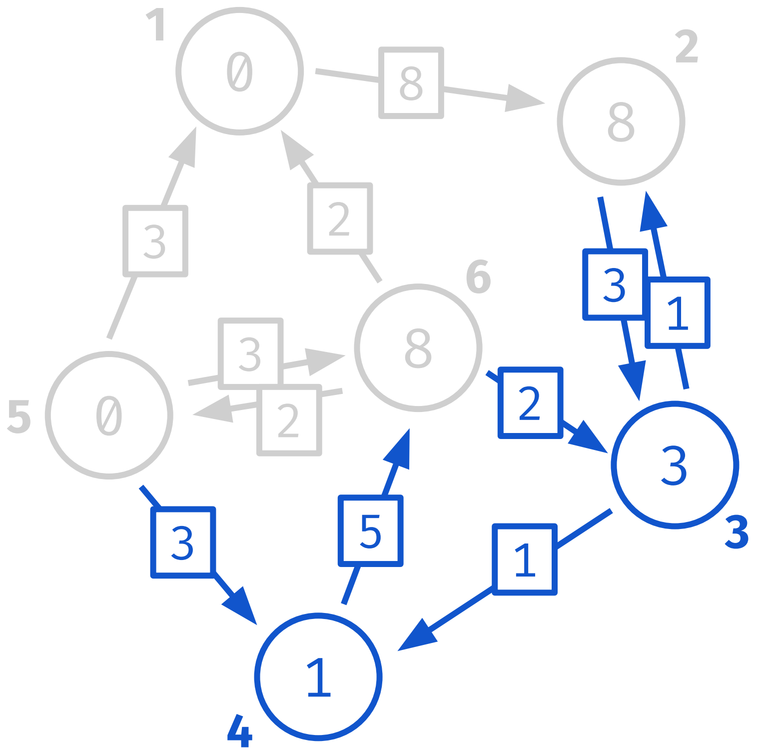
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph

parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

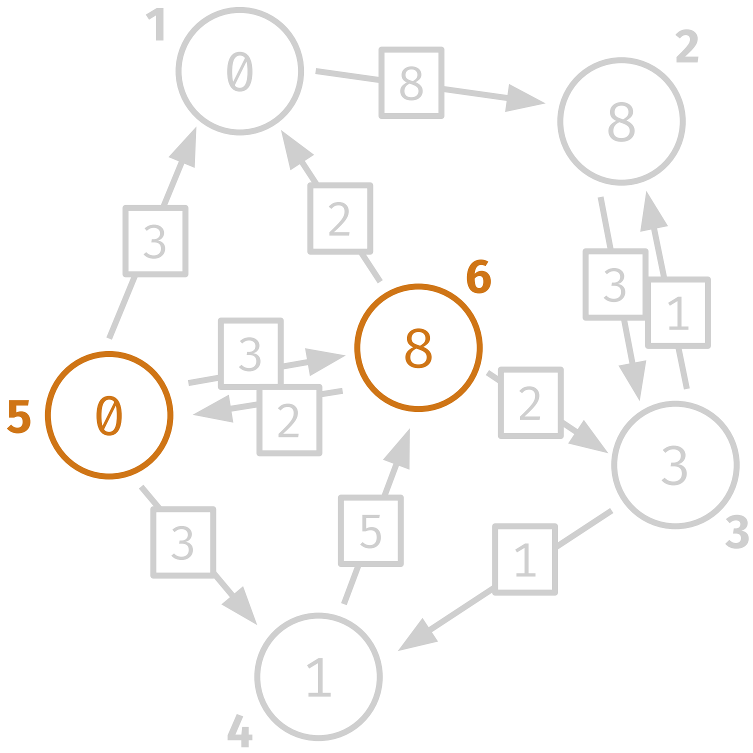
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

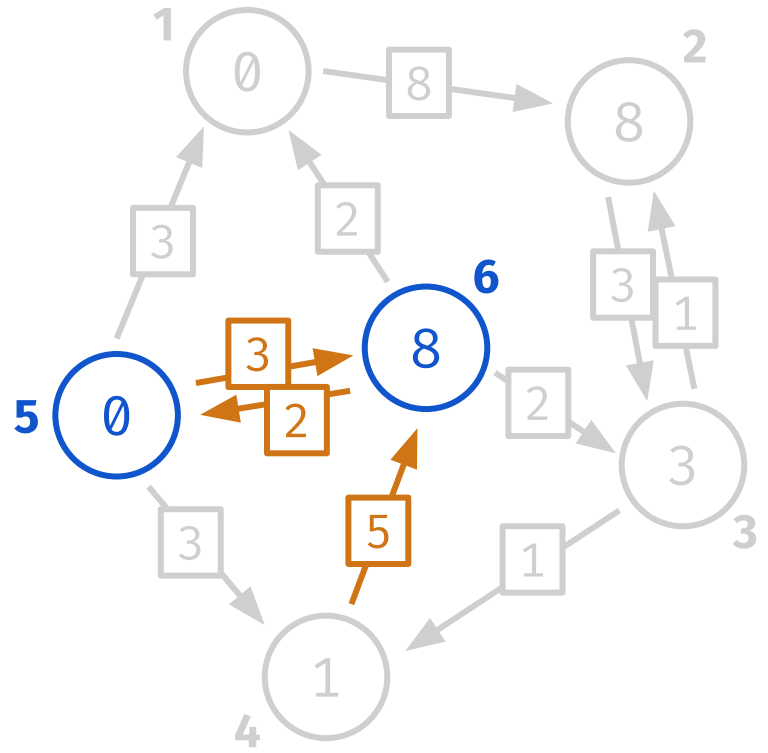
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

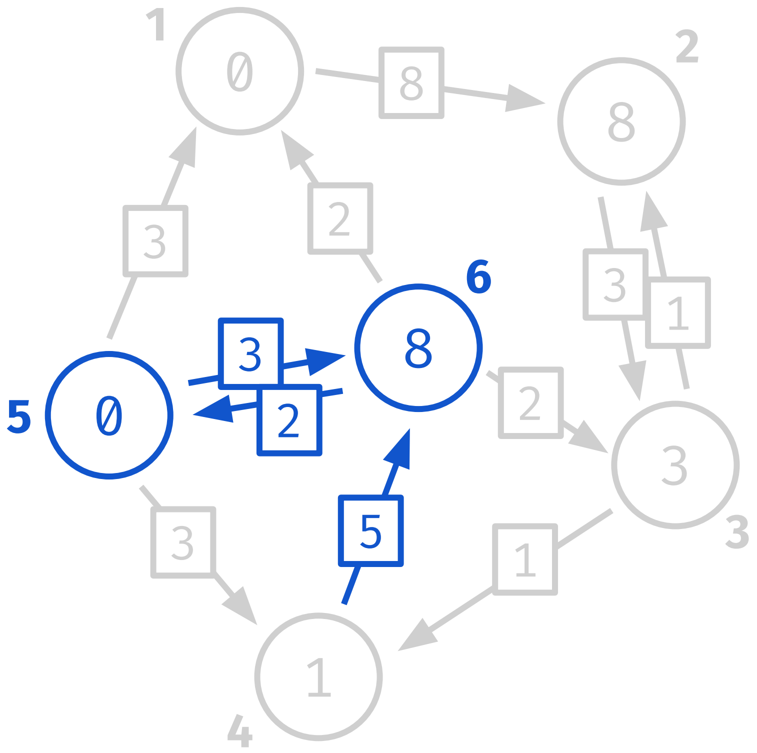
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

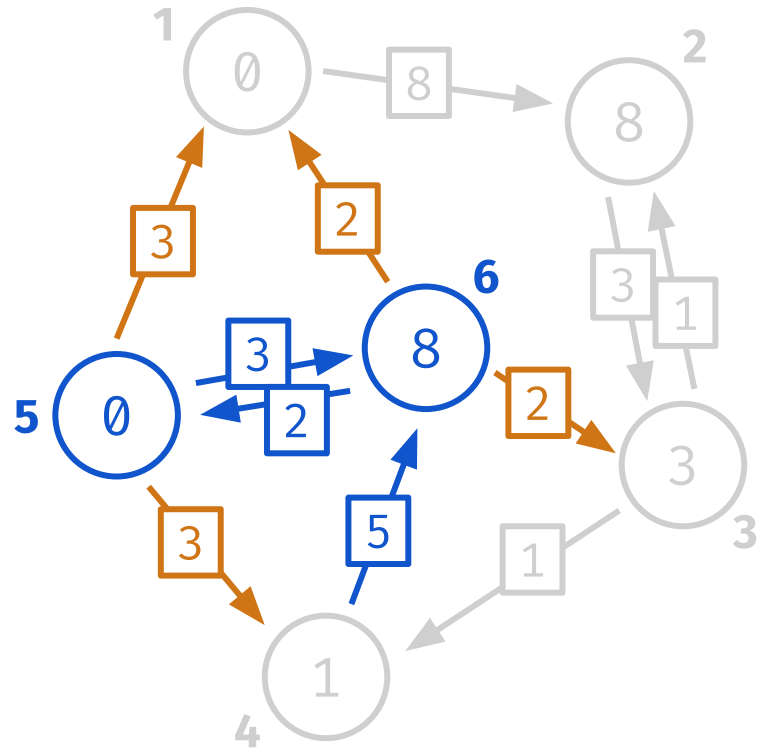
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



parallel sliding window

vertices 1–2

src	dst	val
1	2	8
3	2	1
5	1	3
6	1	2

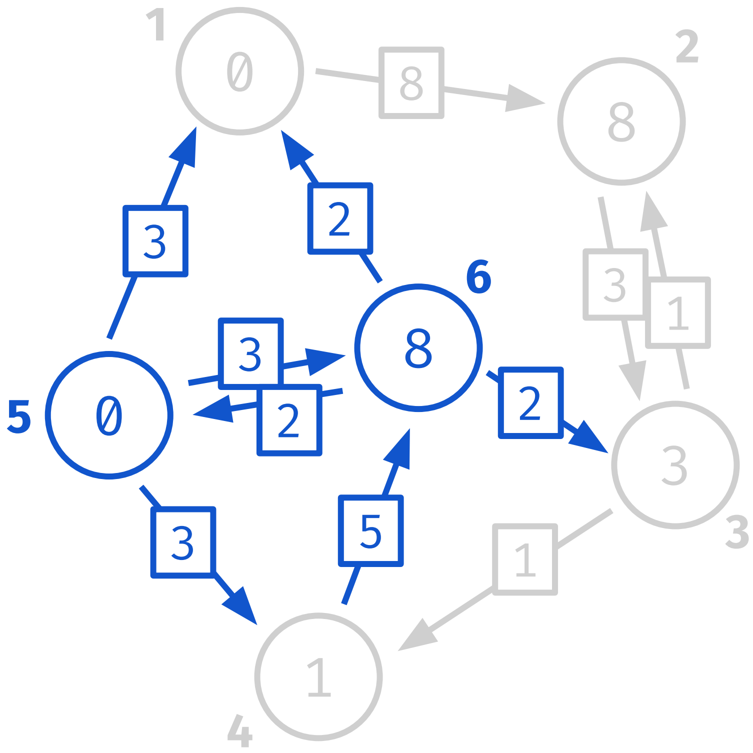
vertices 3–4

src	dst	val
2	3	3
3	4	1
5	4	3
6	3	2

vertices 5–6

src	dst	val
4	6	4
5	6	3
6	5	2

1. load vertices
2. load inedges
3. slide windows
4. load outedges
5. update values
6. write subgraph



size limitations

access pattern speed

parallel sliding window

other things i didn’t have time to talk about

- the IO cost of PSW
- how PSW can handle adding and removing edges
- PSW is asynchronous and visits vertices in a specific order
  - despite these limitations, still has good applications

GraphChi: Large-Scale Graph Computation on Just a PC

Aapo Kyrölä, Guy Blelloch, Carlos Guestin

CJ QUINES