Graph Algorithms in a Guaranteed-Deterministic Language

Praveen Narayanan & Ryan R. Newton

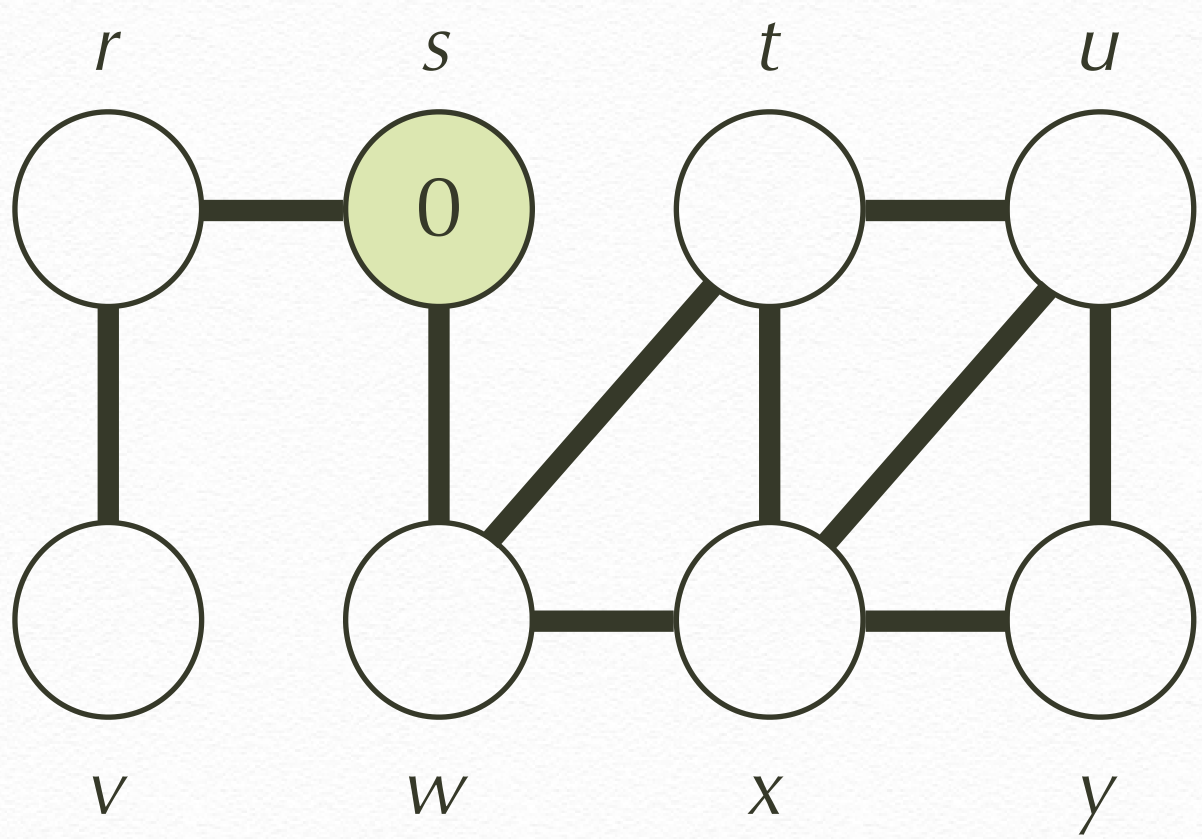
Outline

- An example of non-determinism
- Introduction to LVish
- BFS and MIS in LVish
- BulkRetry, a new addition

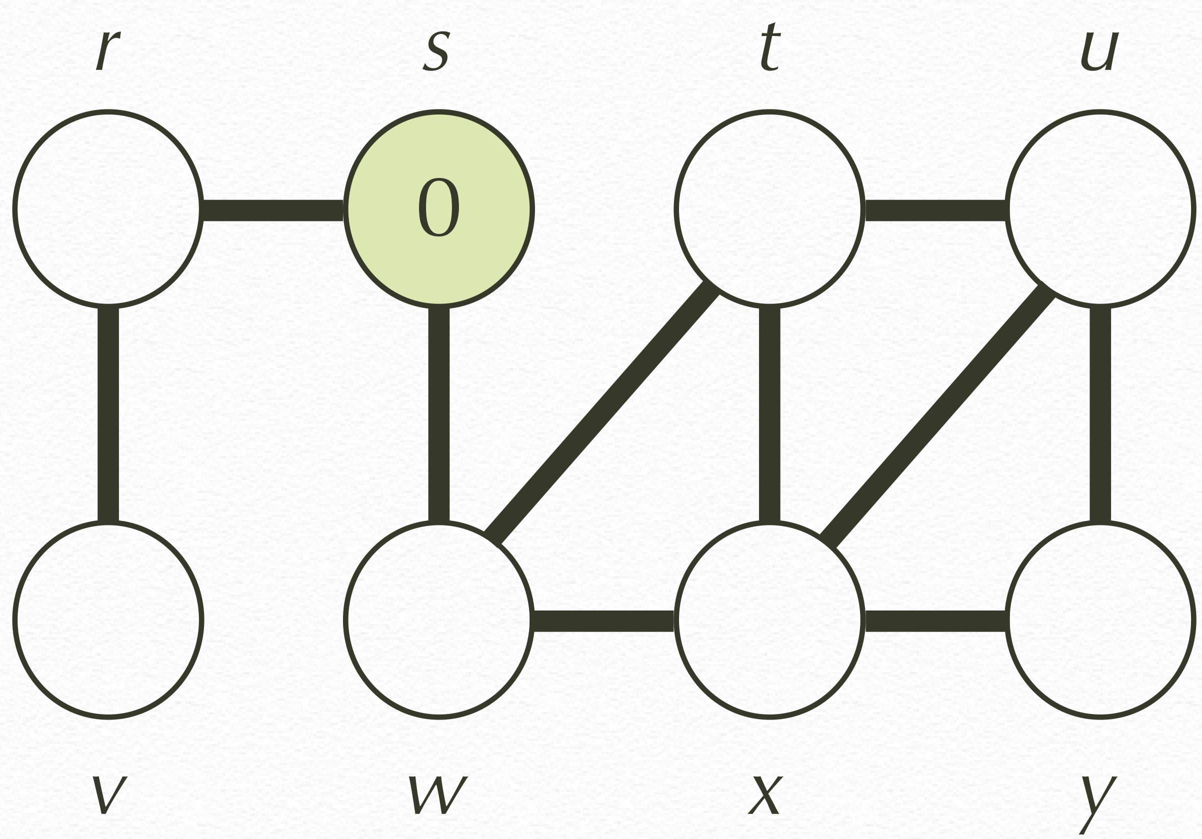
Graph algorithms

- *Irregular* parallel behavior
- Amount of parallelism depends on input

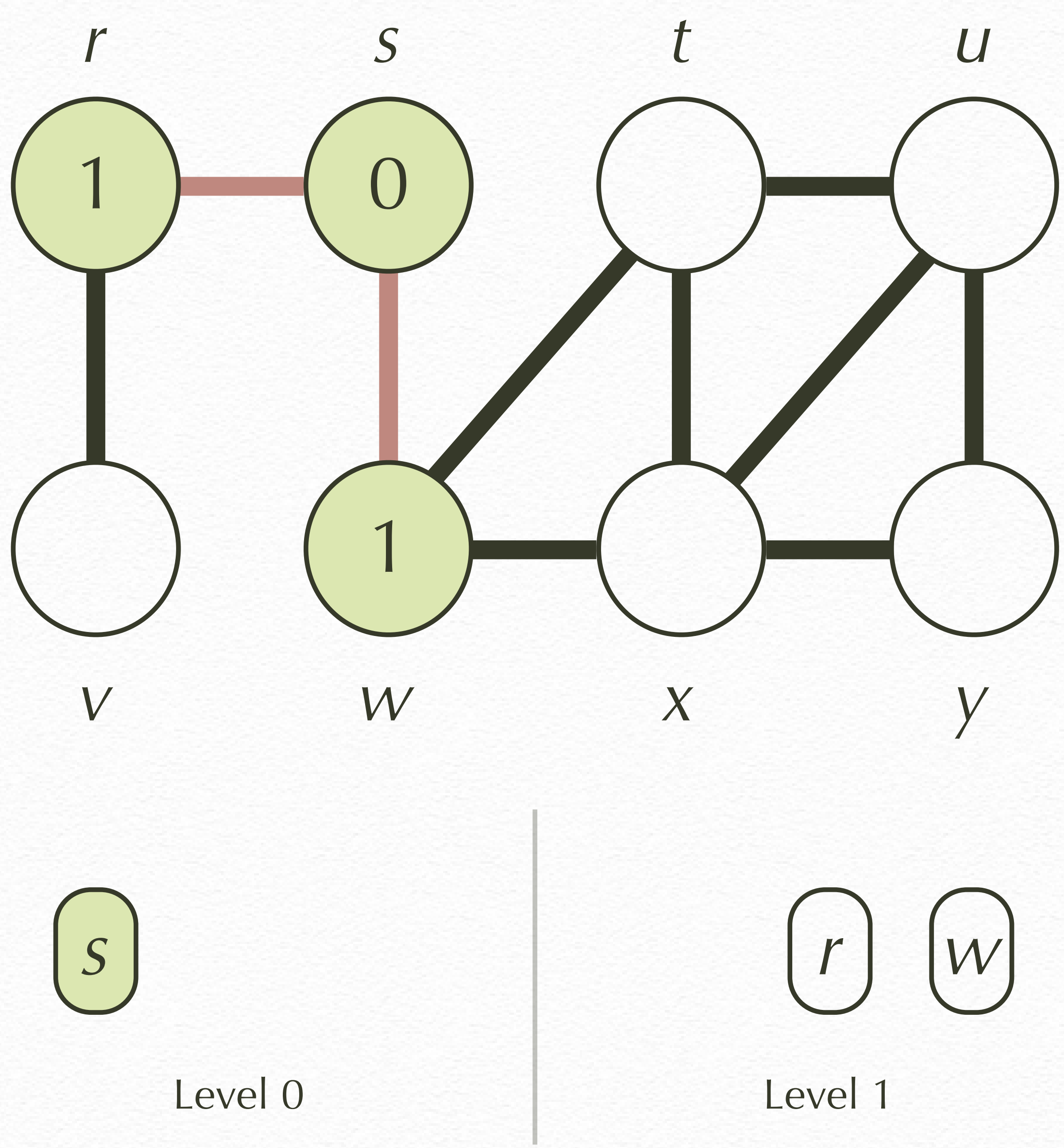
Parallel BFS



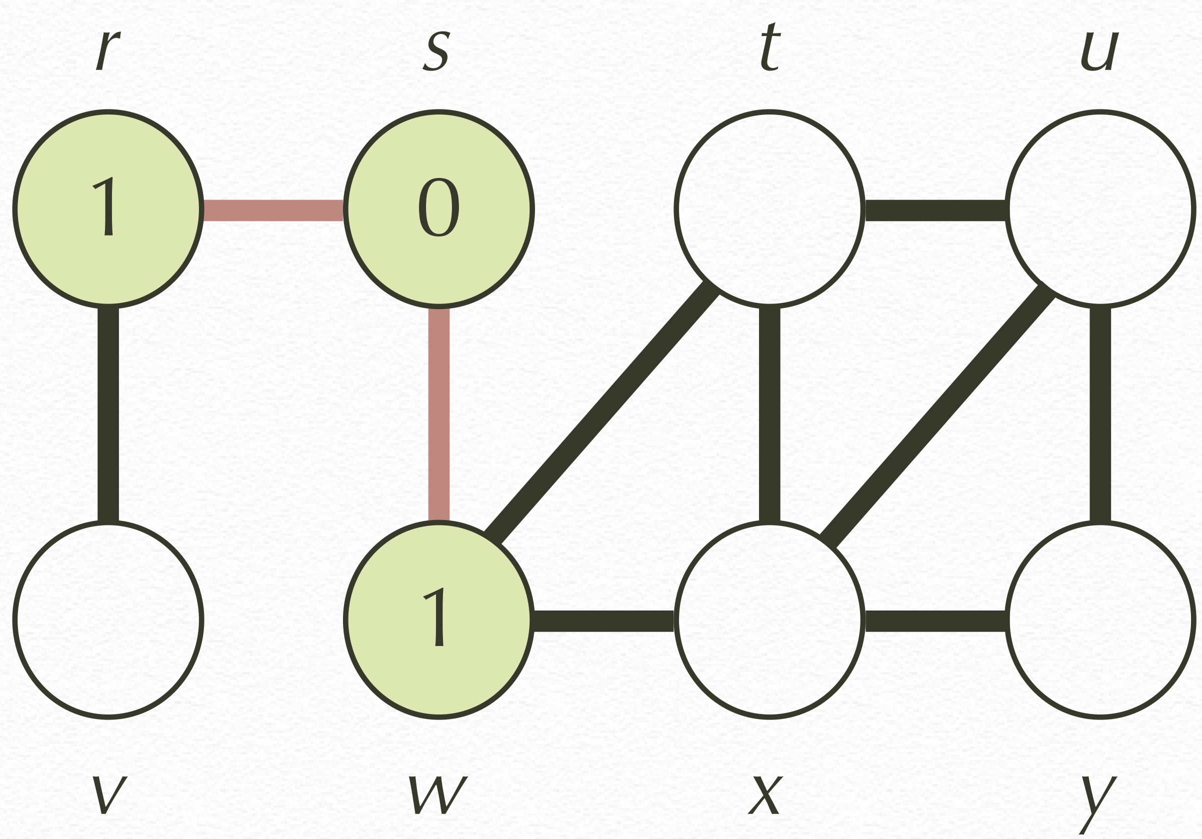
Parallel BFS

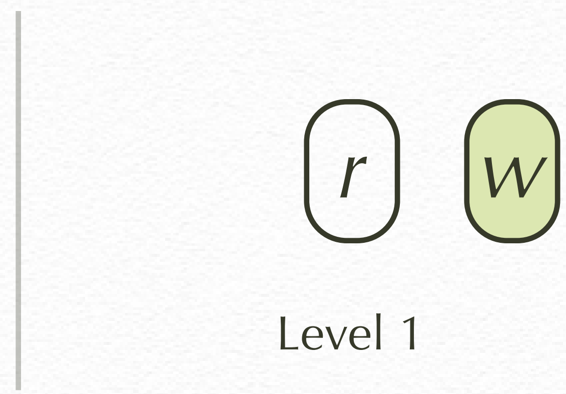


Parallel BFS

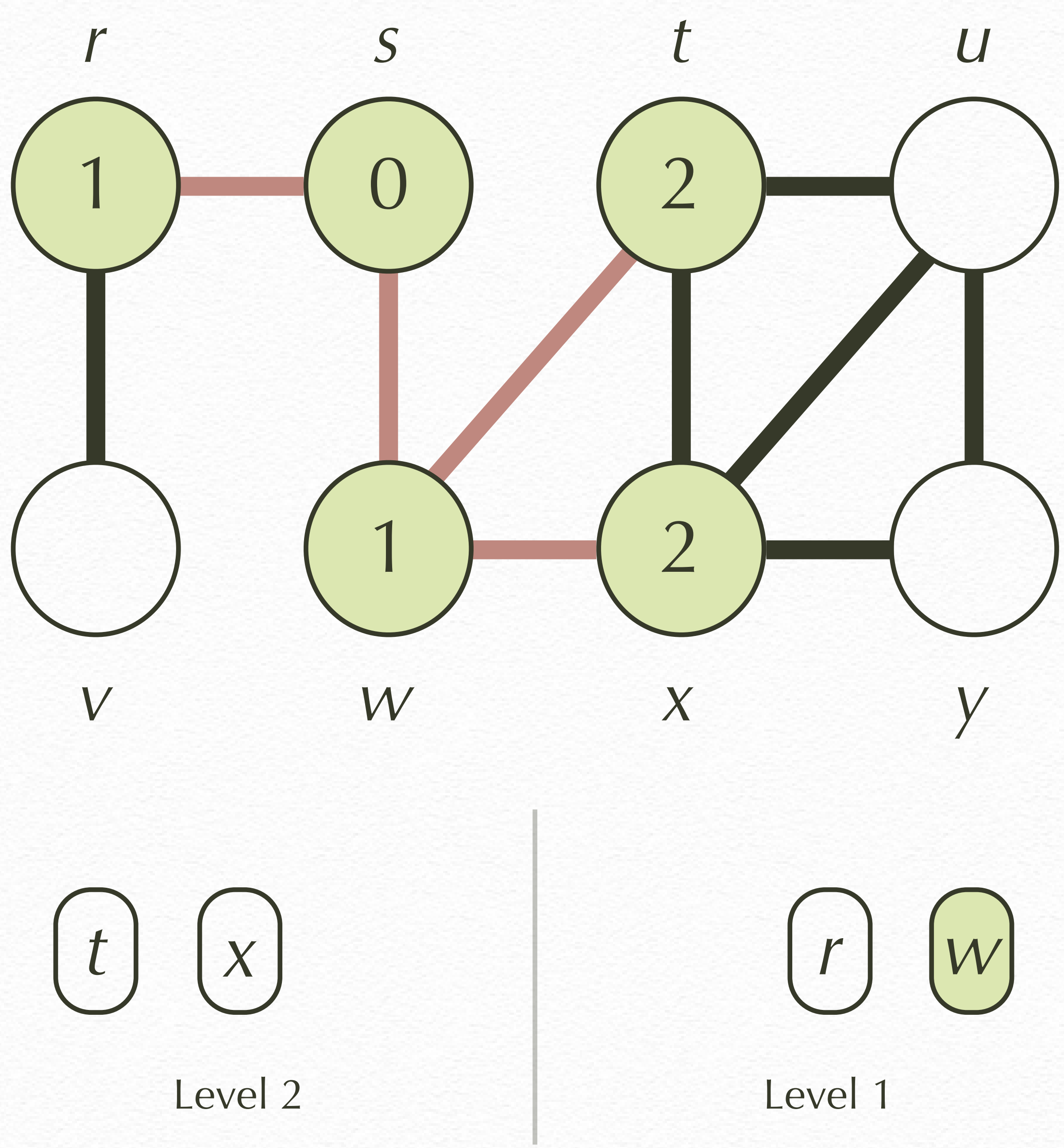


Parallel BFS

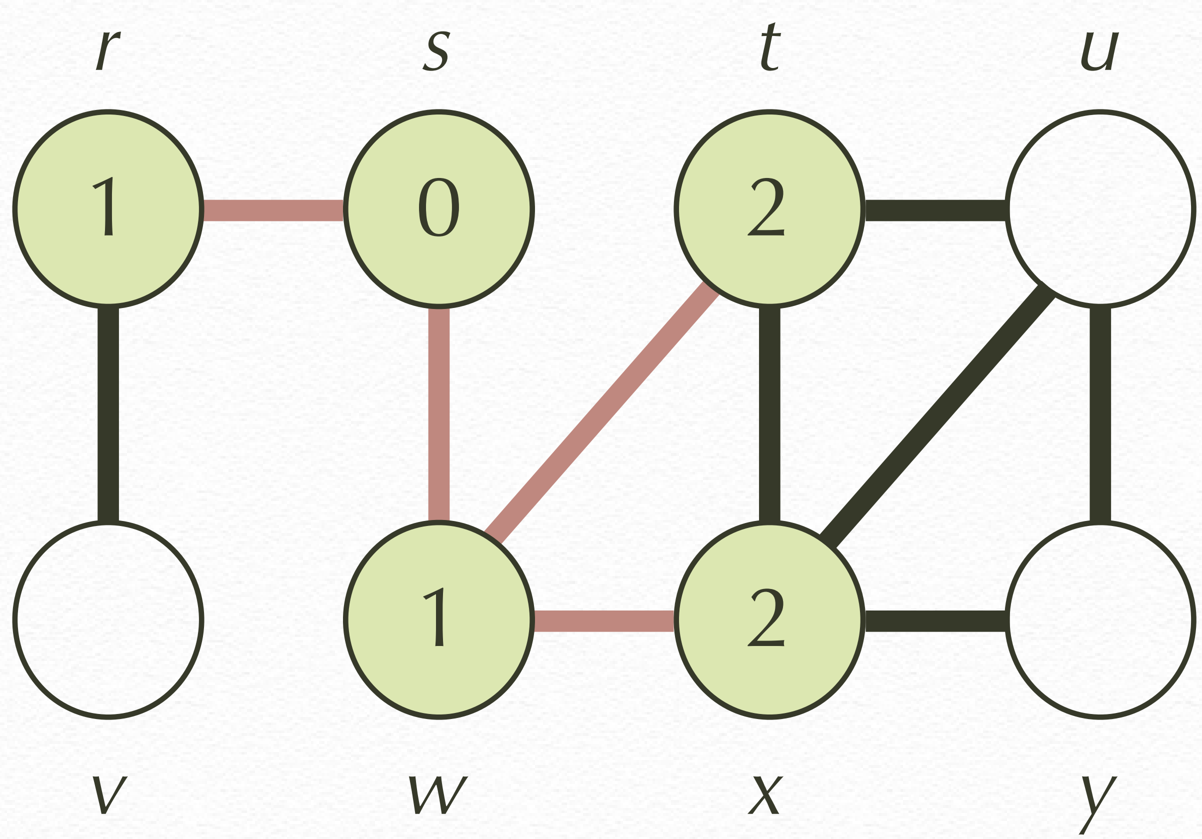




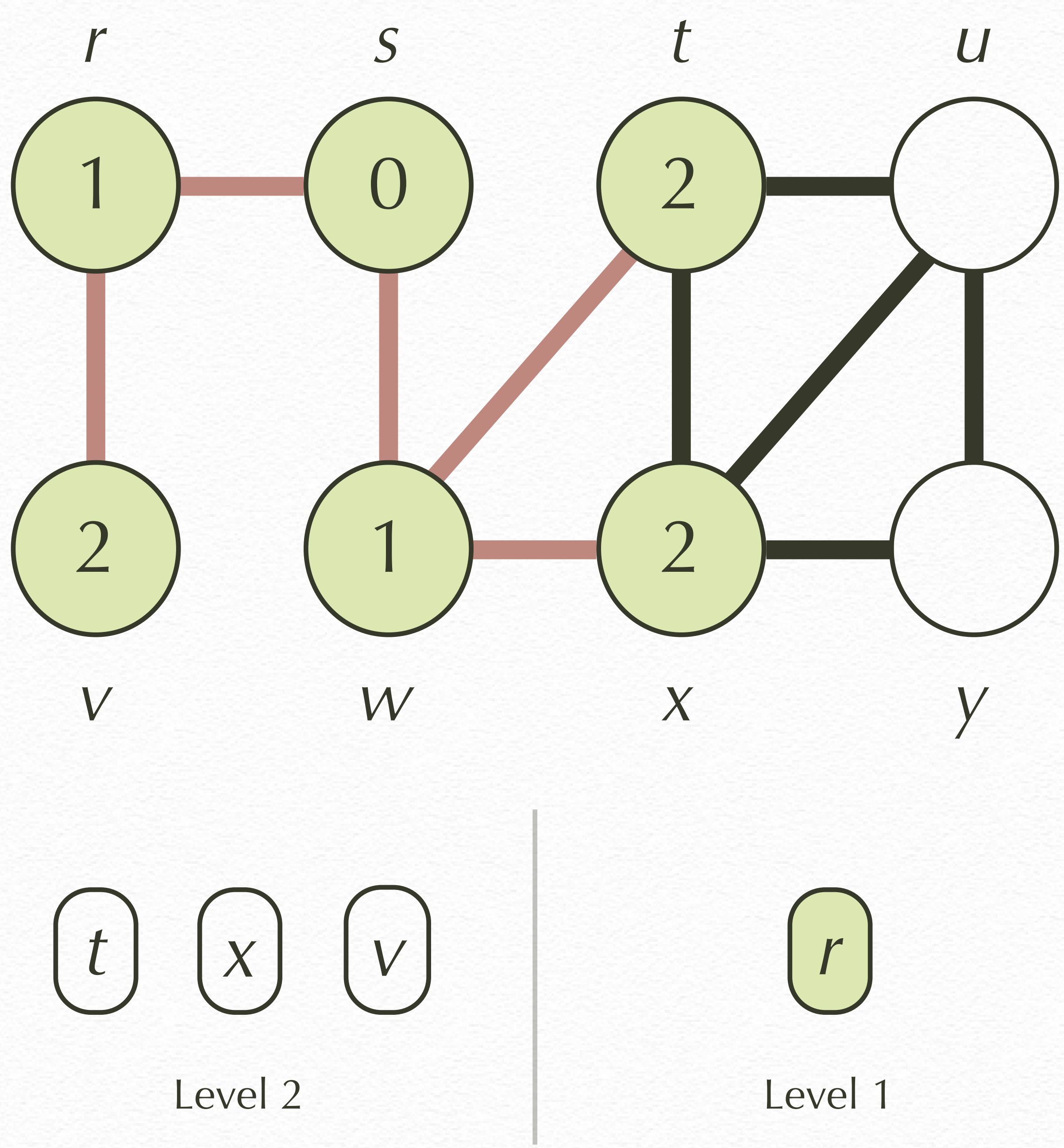
Parallel BFS



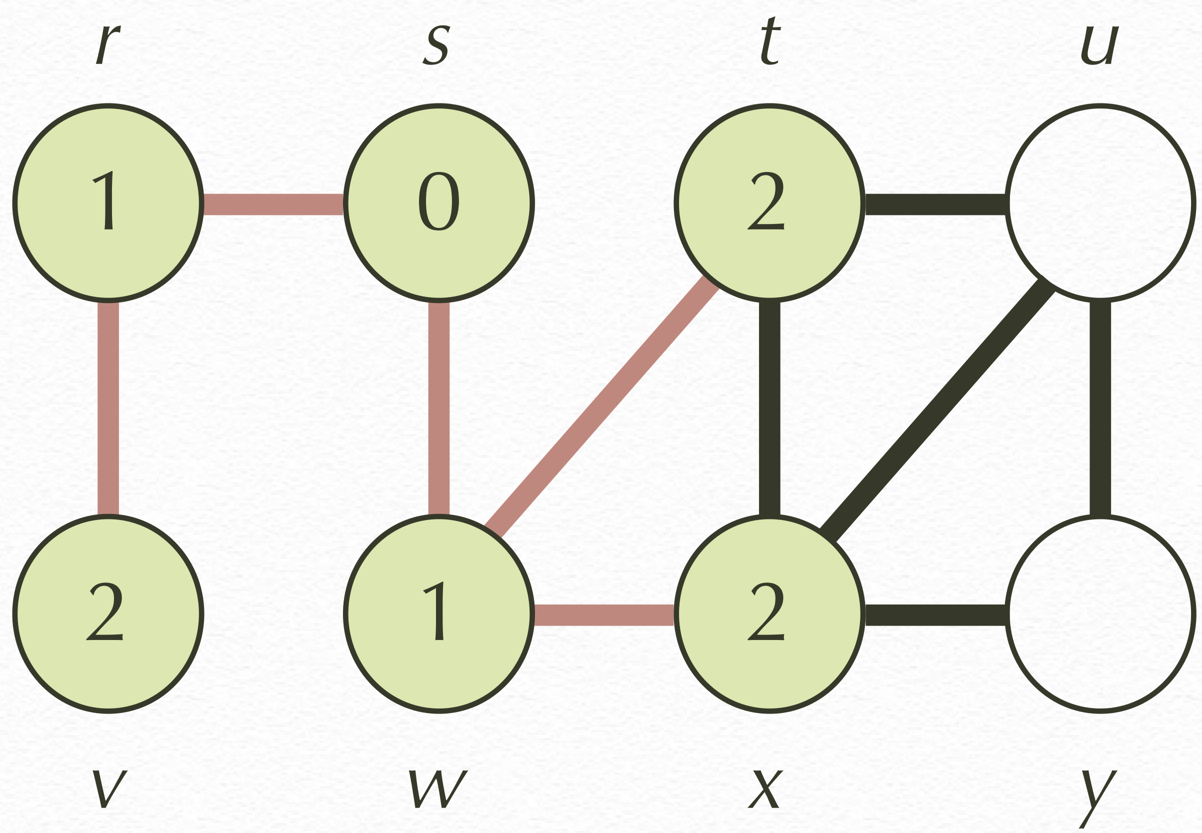
Parallel BFS

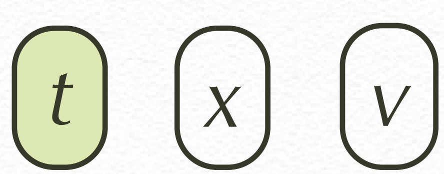


Parallel BFS



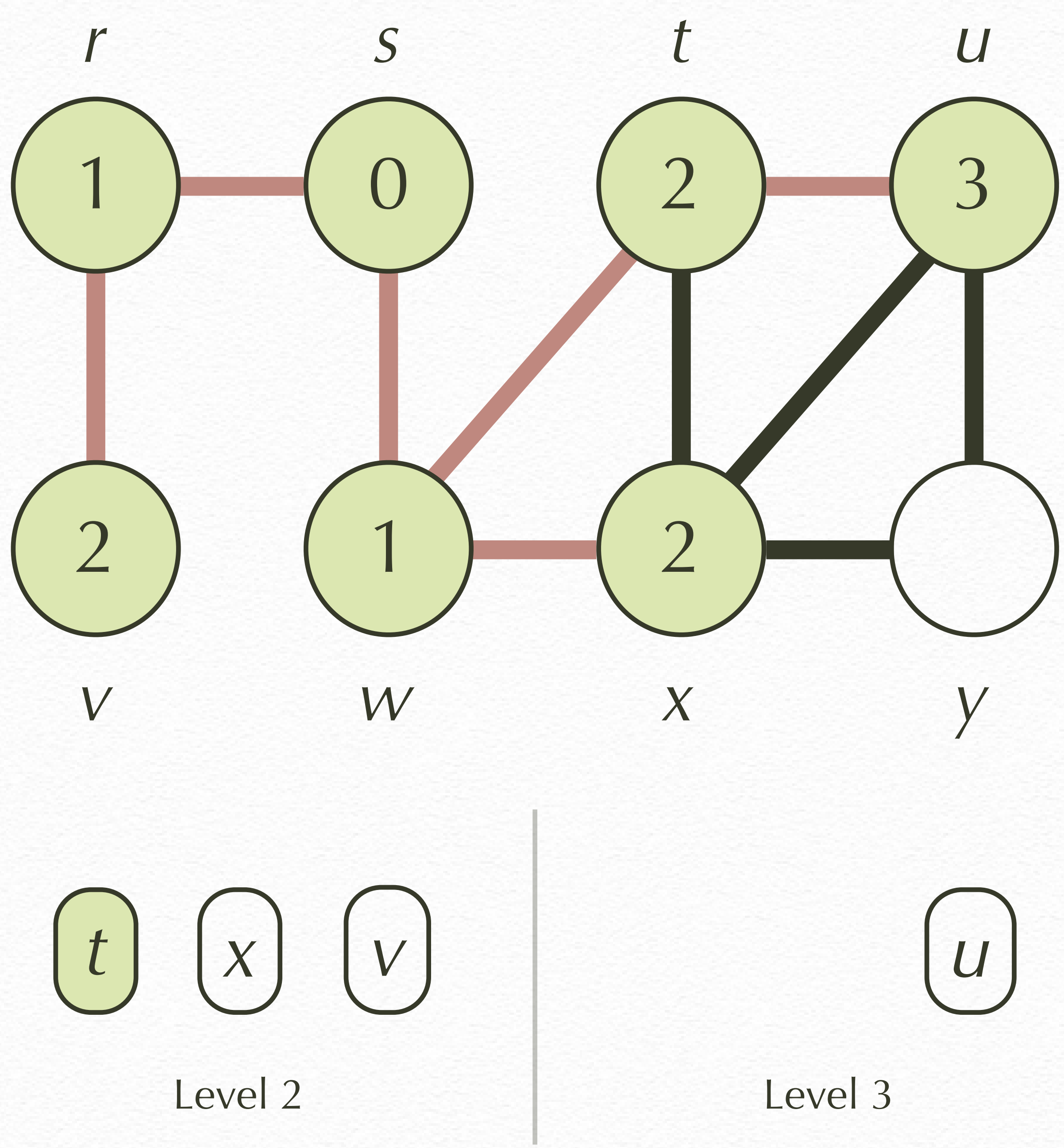
Parallel BFS



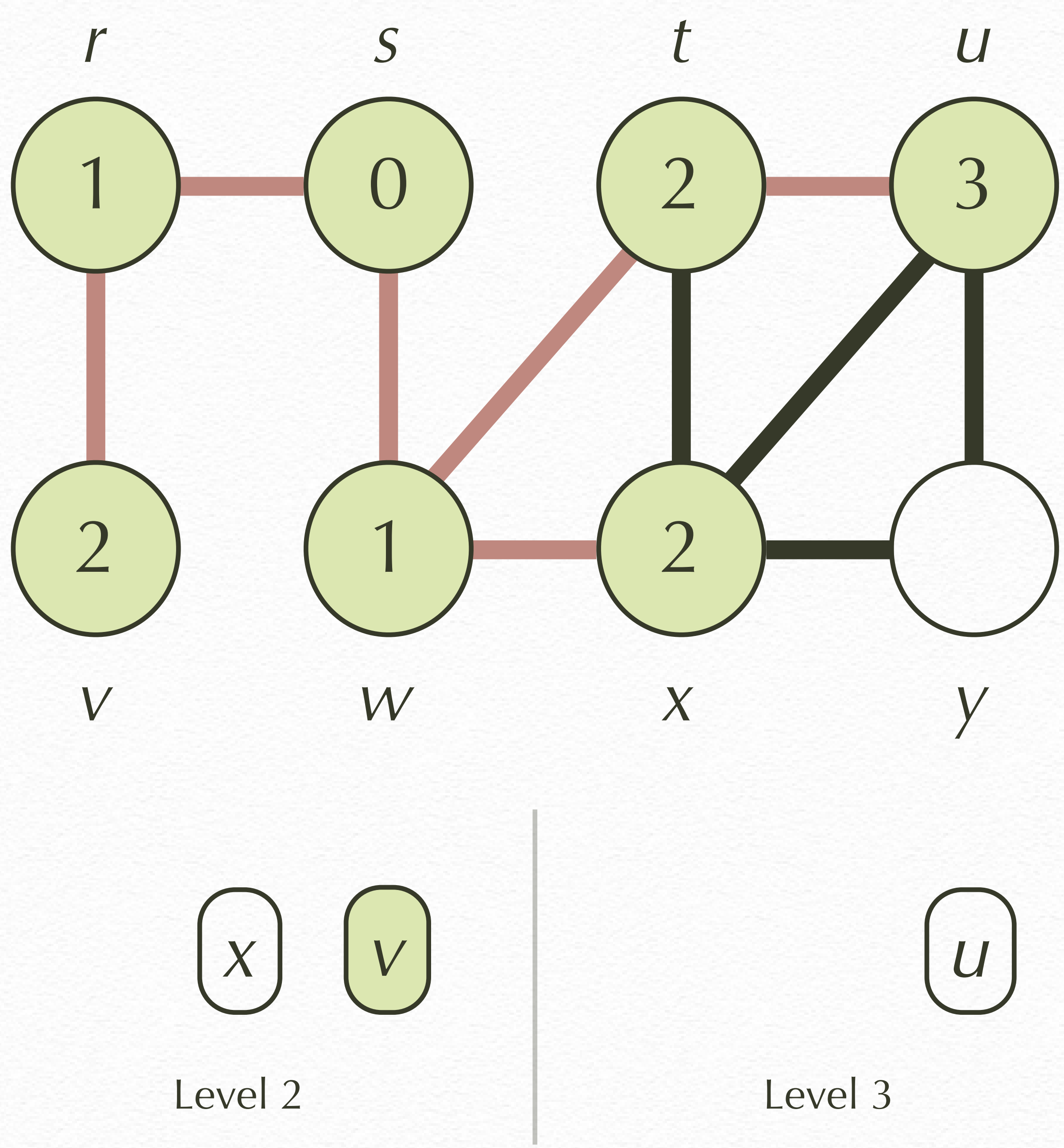


Level 2

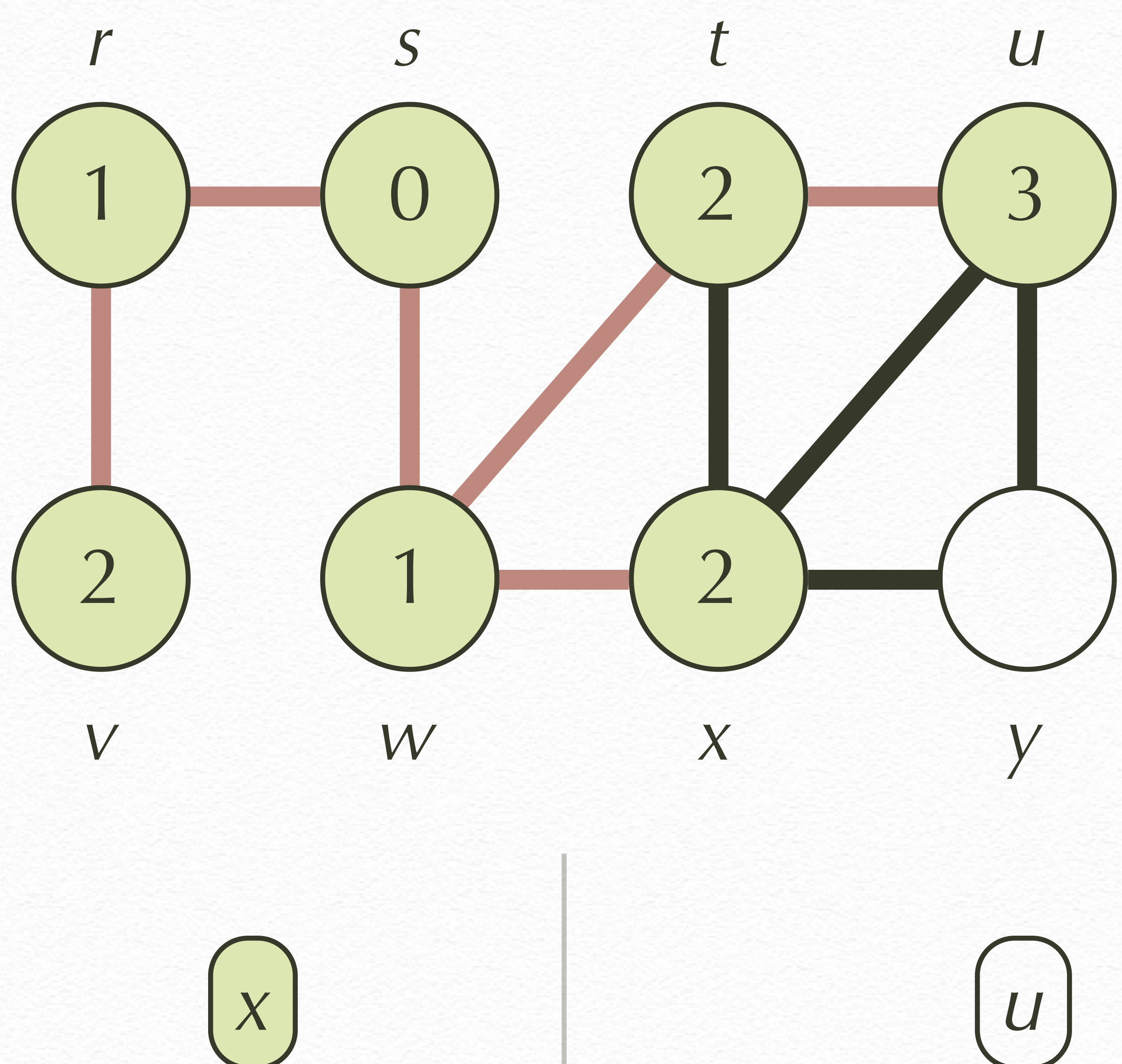
Parallel BFS



Parallel BFS



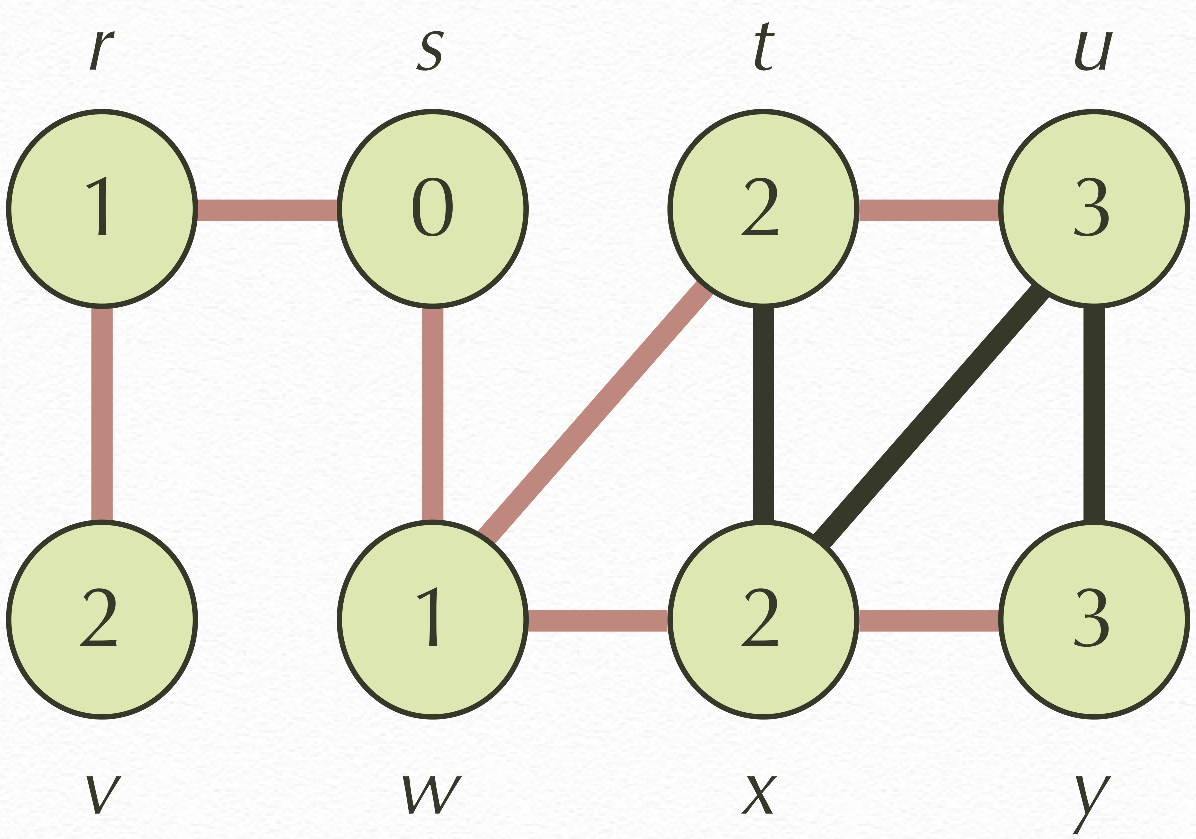
Parallel BFS

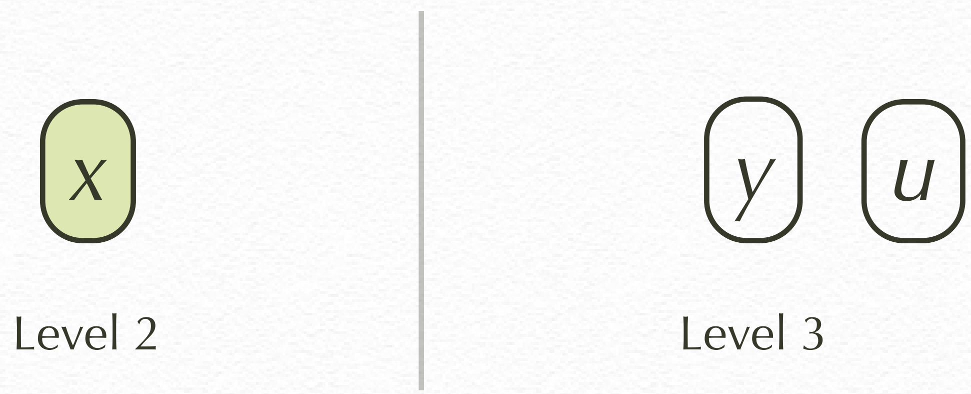


Level 2

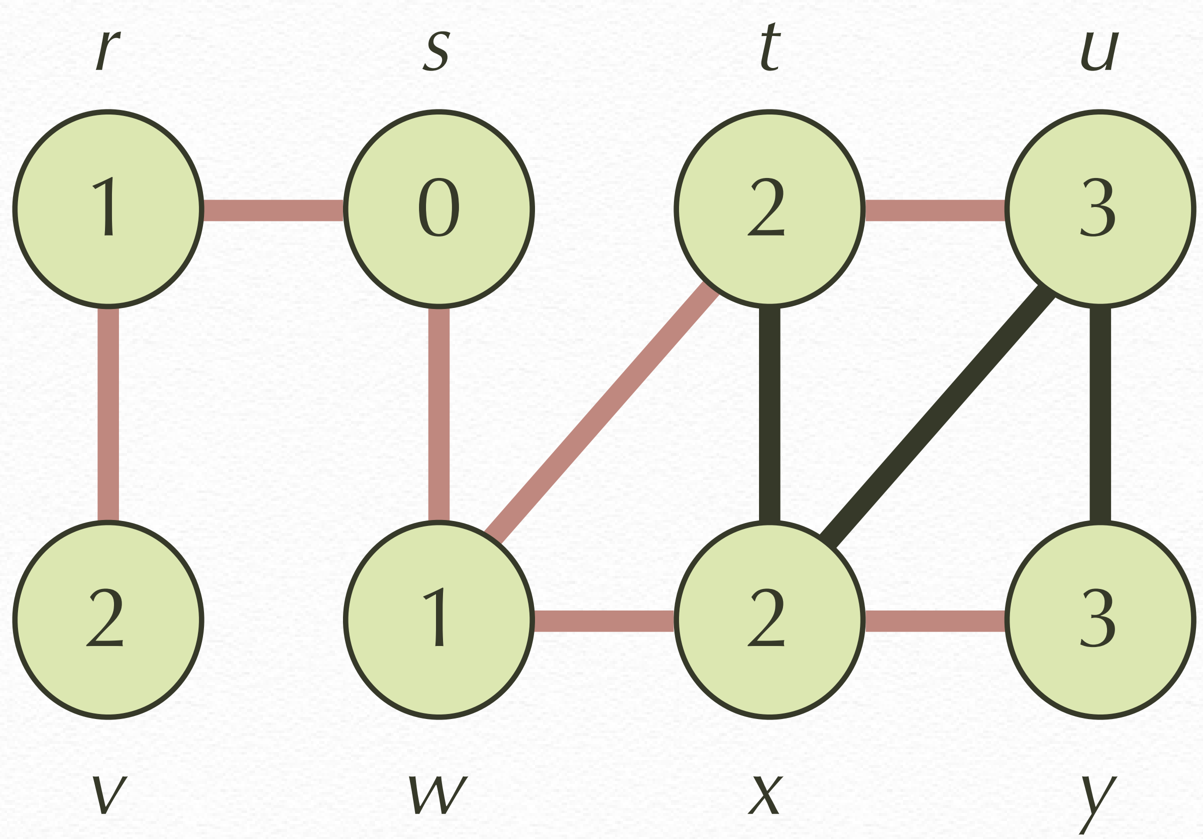
Level 3

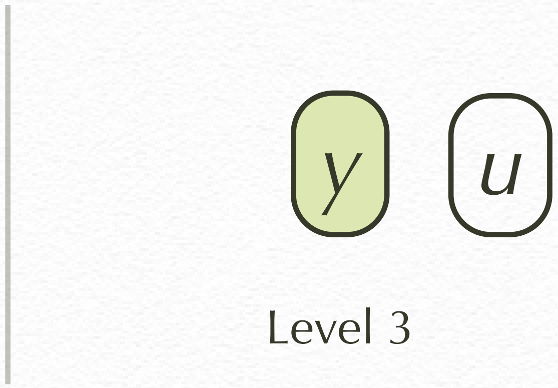
Parallel BFS



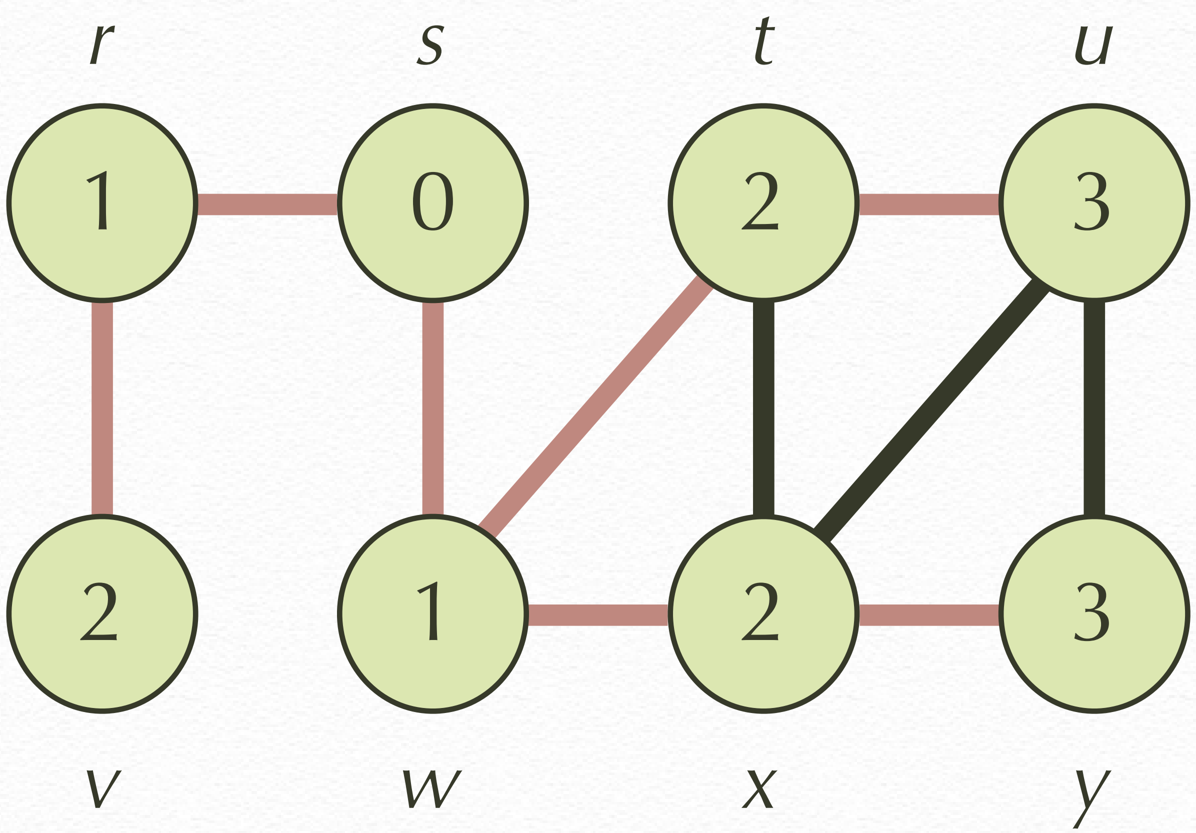


Parallel BFS

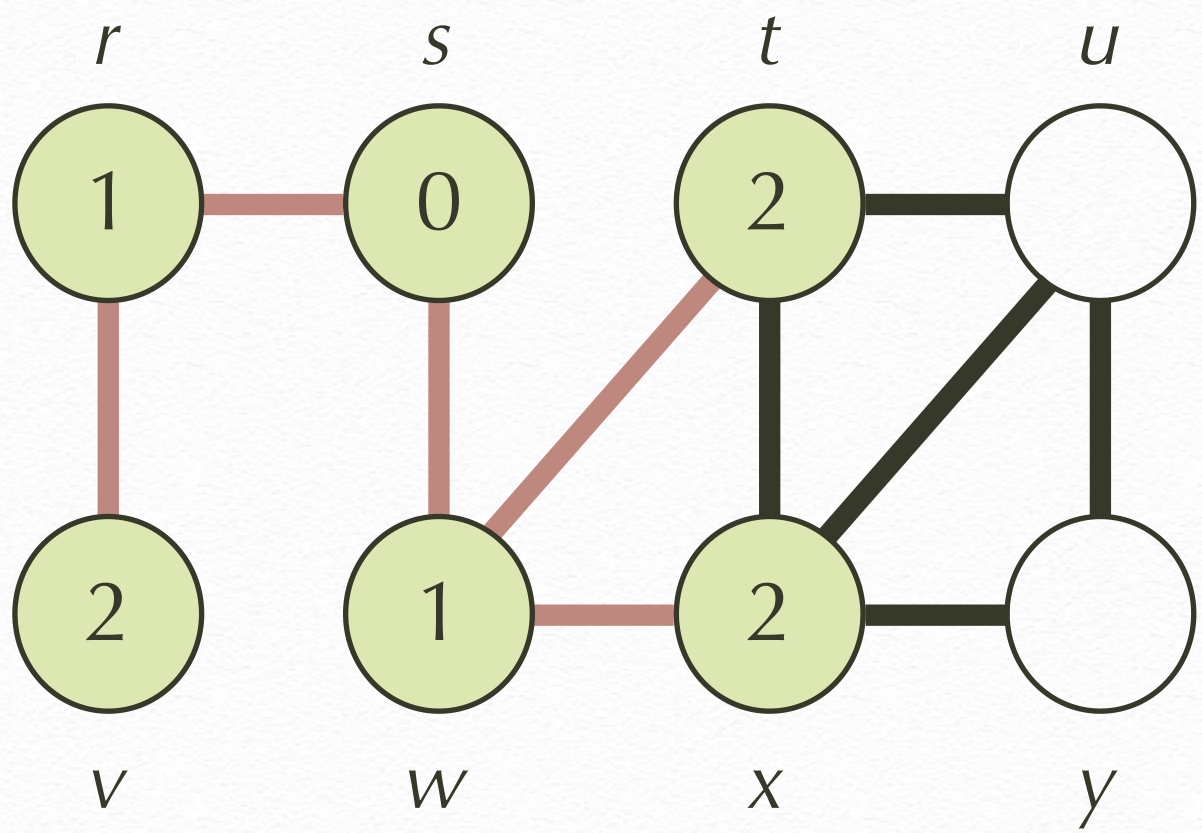


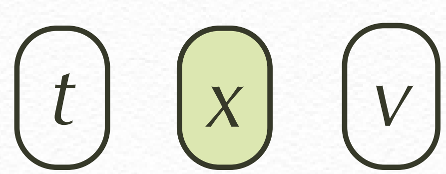


Parallel BFS



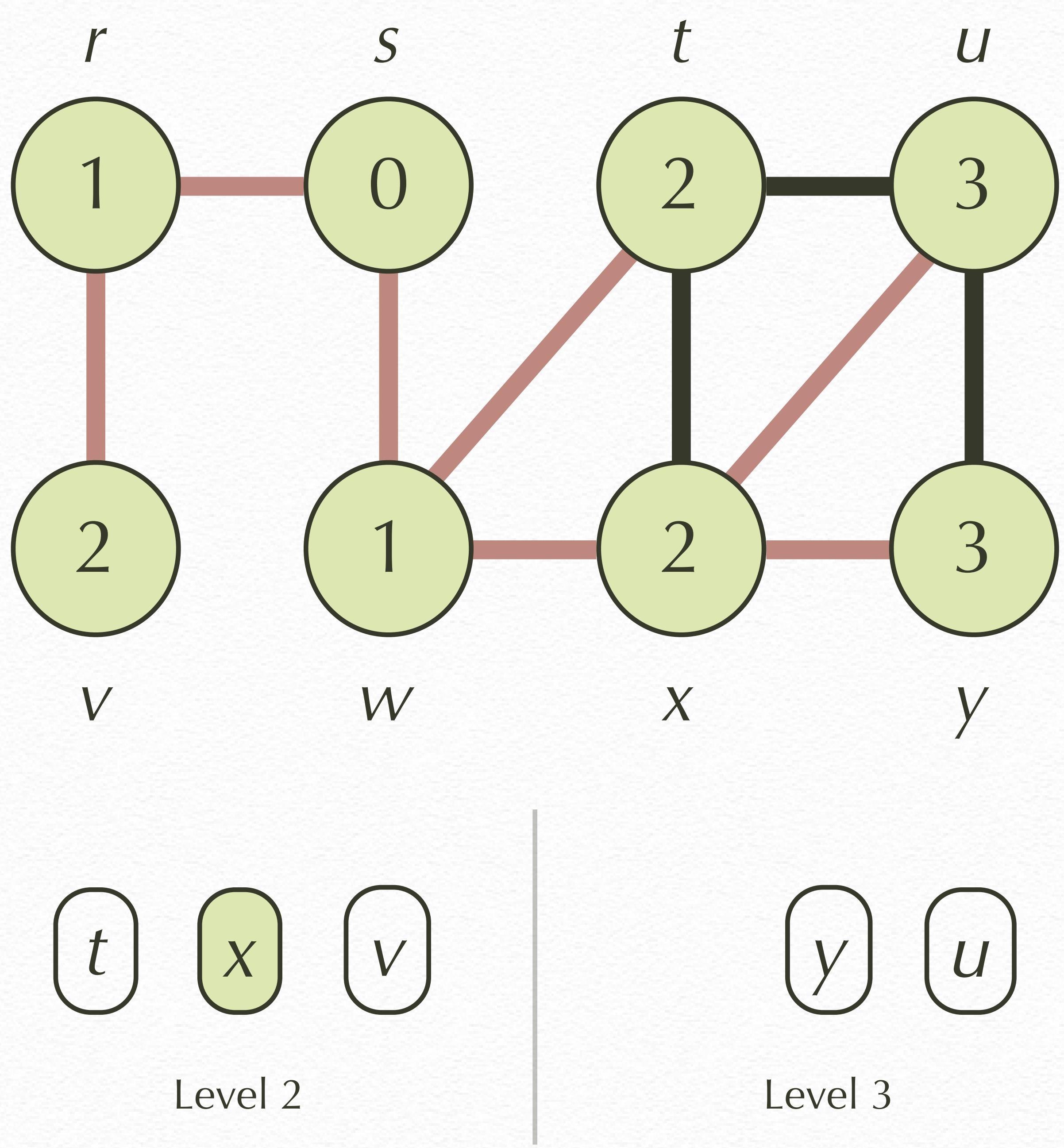
Choose a different winner



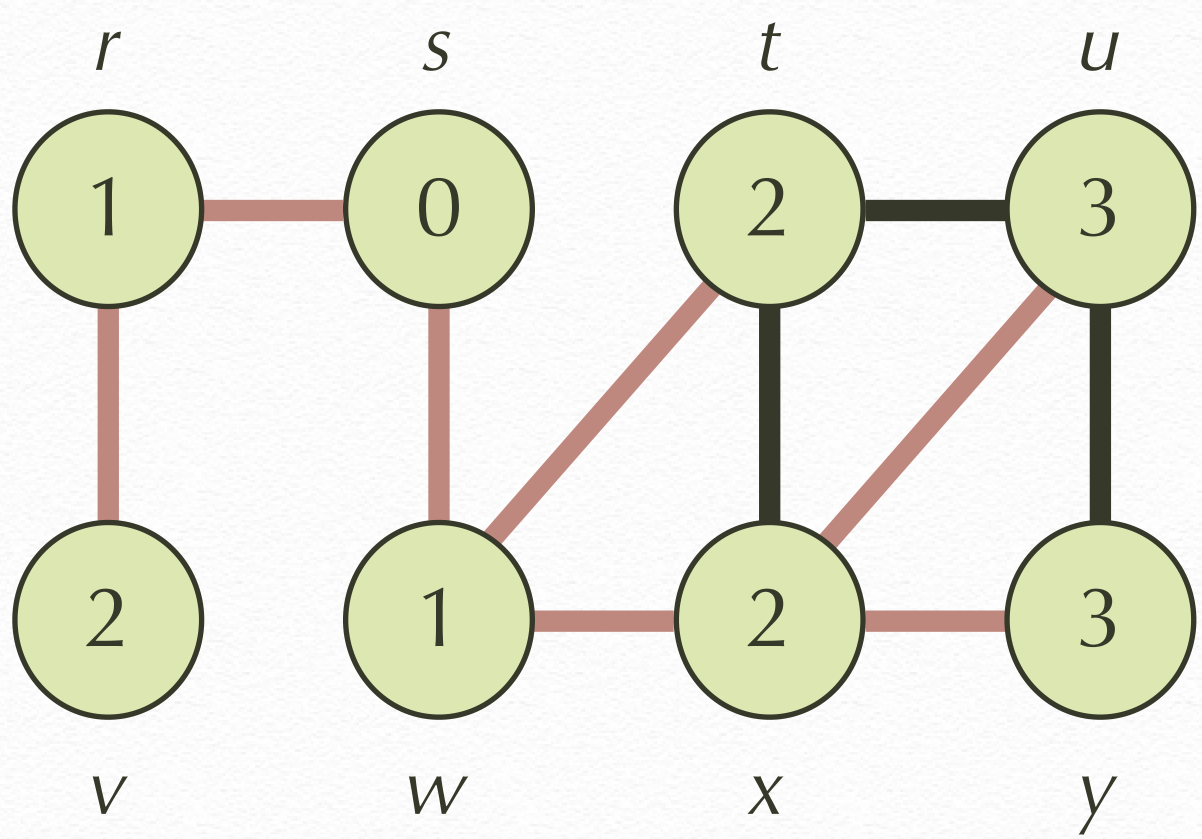


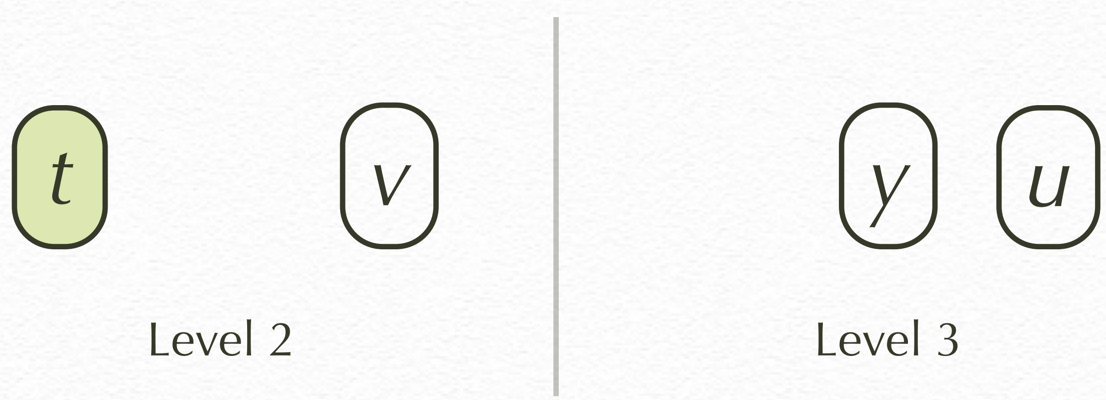
Level 2

Different BFS tree

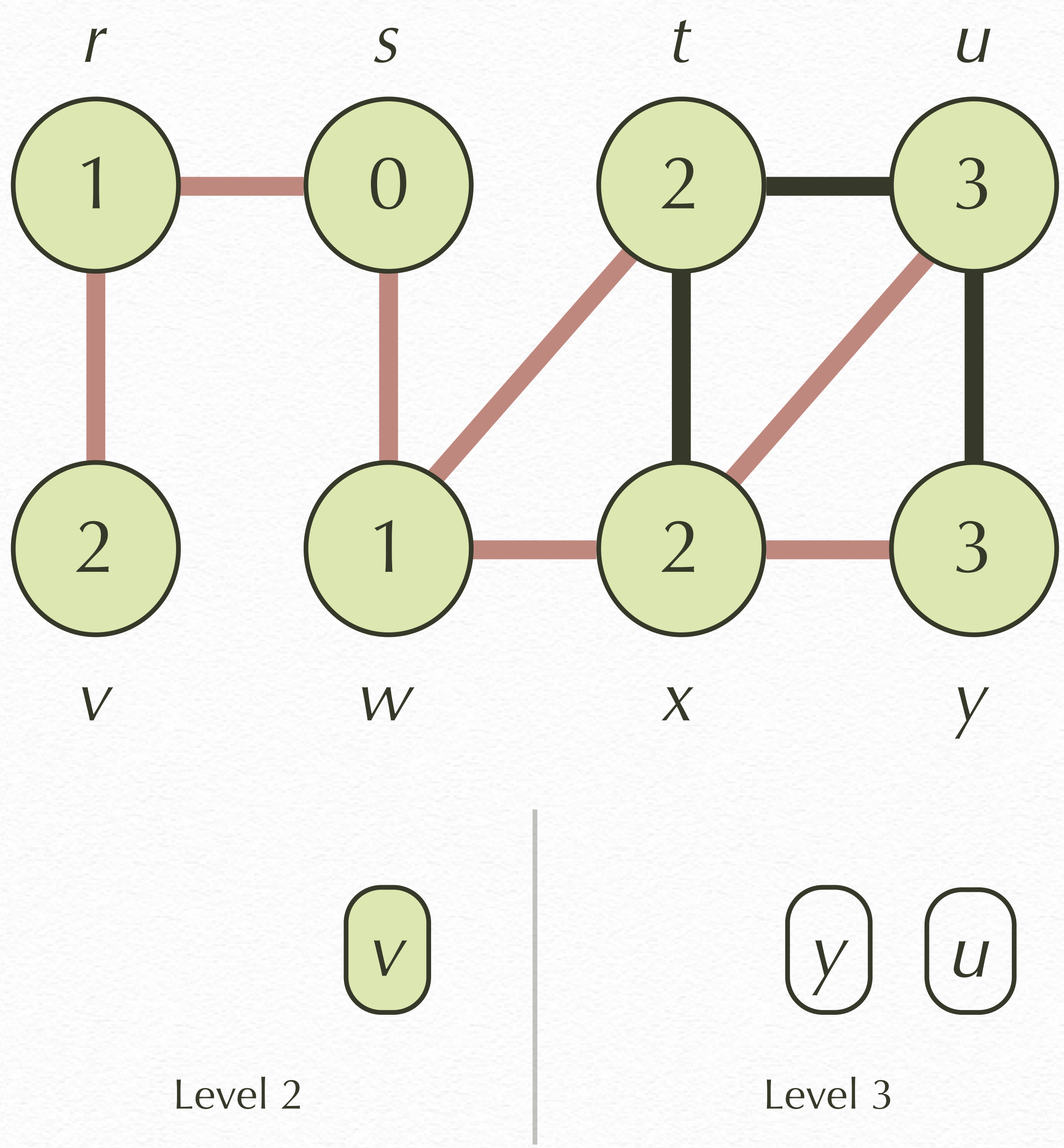


Different BFS tree

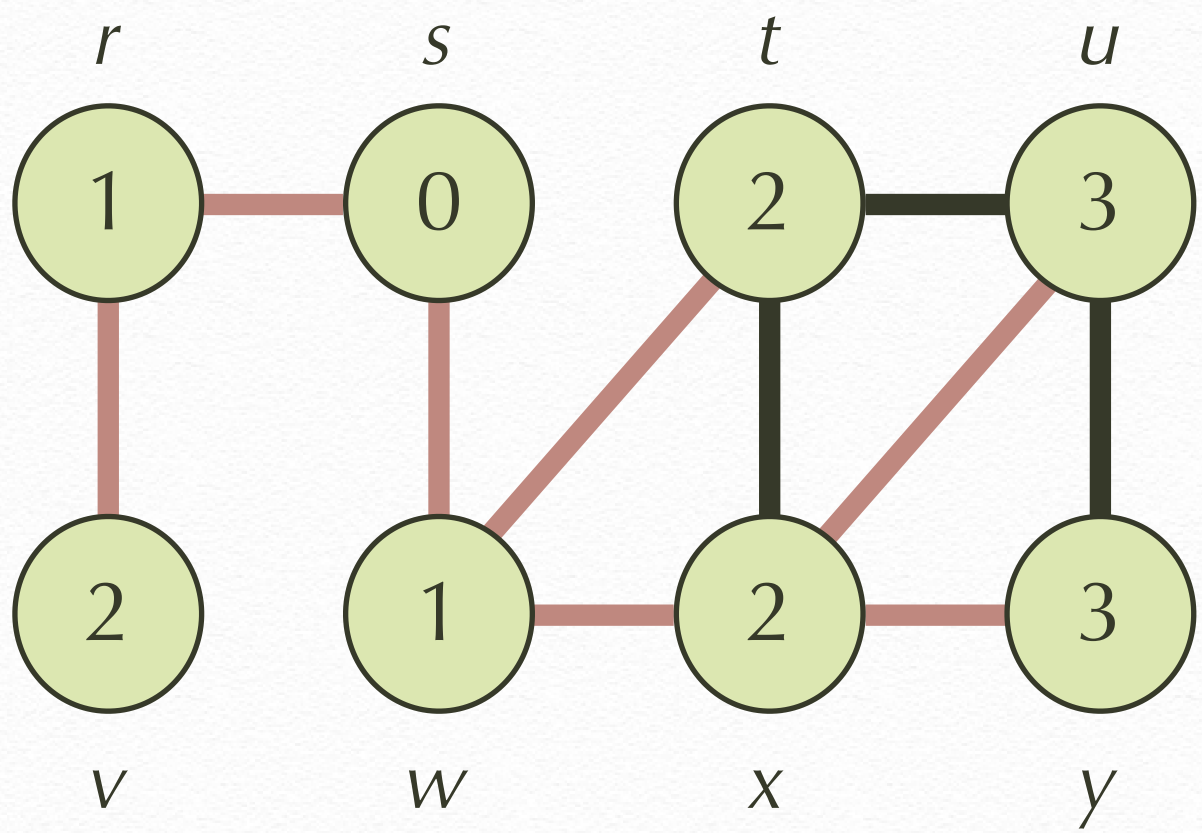


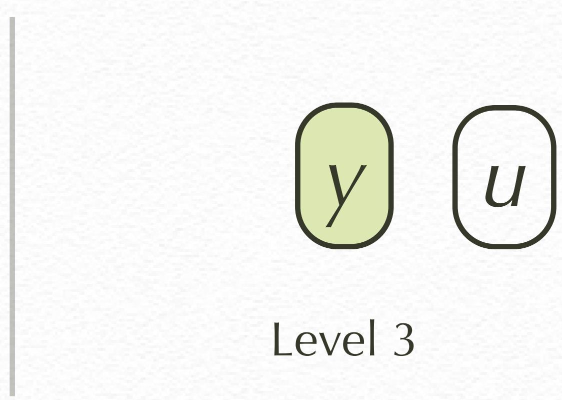


Different BFS tree

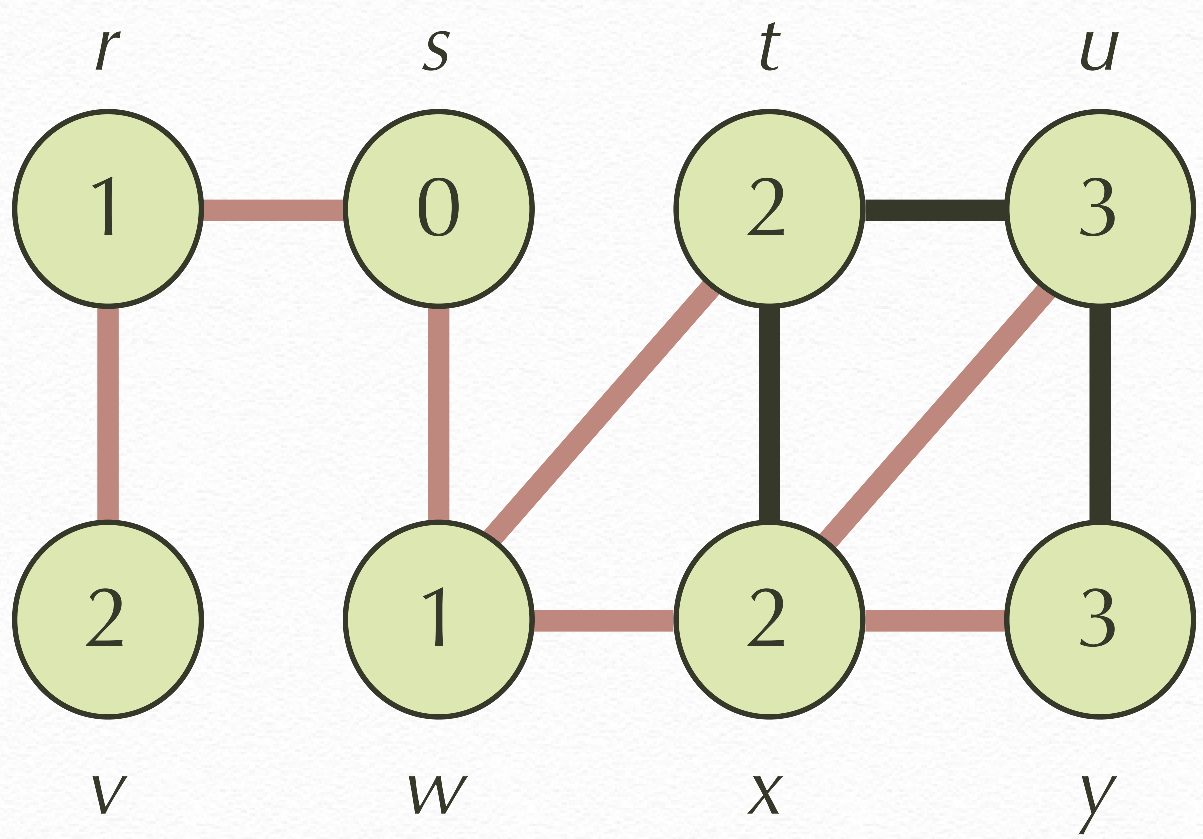


Different BFS tree

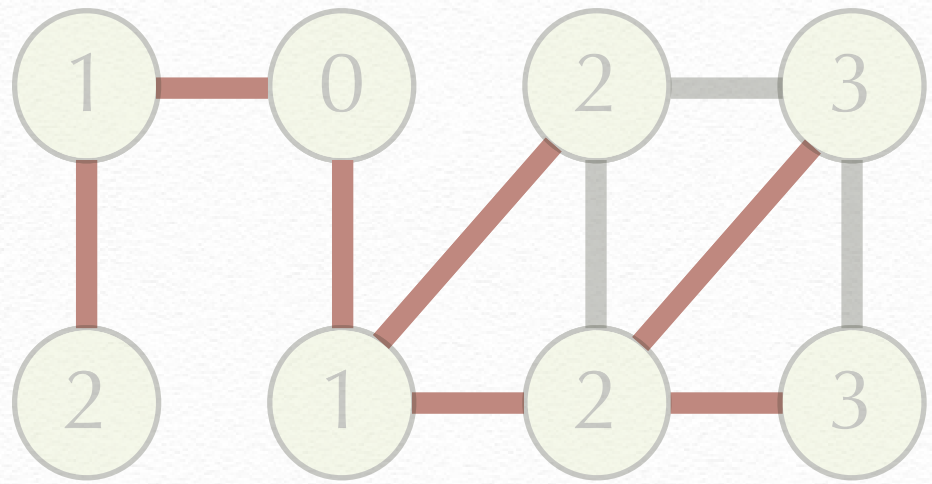


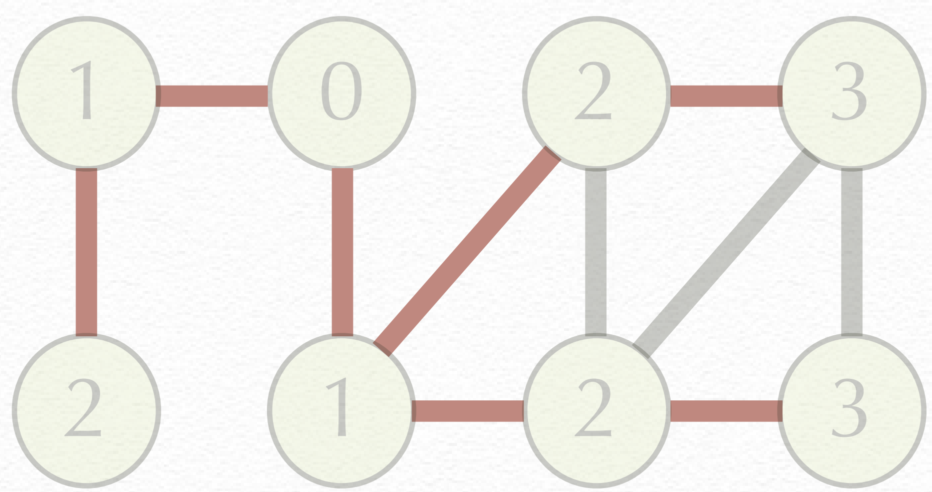


Different BFS tree



Non-determinism





How could we achieve determinism?

- Carefully write code in a *non-deterministic* language [Blelloch *et al.*, 2012]
- *Program-level* guarantee only

```
node.visited = true
```

How could we achieve determinism?

- Implement algorithms in a *guaranteed-deterministic* language (this talk)

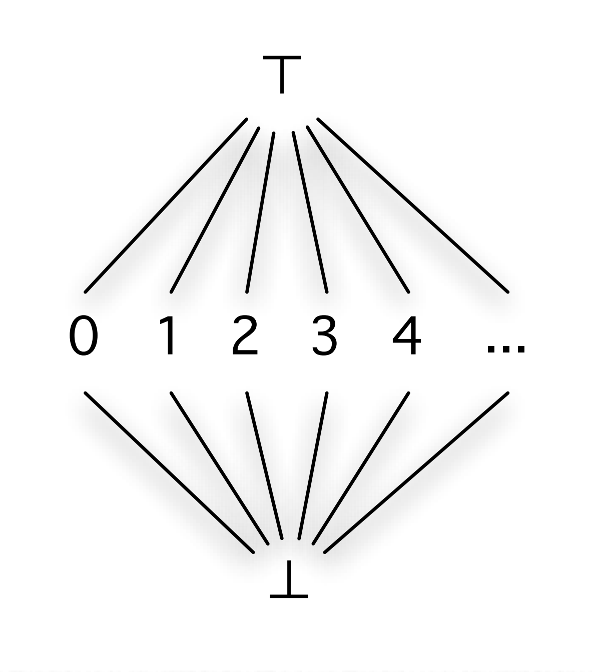
- *Language-level* guarantee

LVars

- Lattice Variables [FHPC '13]
- *Monotonic* data structures

LVars

- Multiple *least-upper-bound* writes



Lattice for `num`

Raises an error, since $3 \sqcup 4 = \top$

```
do
  fork (put num 3)
  fork (put num 4)
```

Fine, since $4 \sqcup 4 = 4$

```
do
  fork (put num 4)
  fork (put num 4)
```

LVars

- Blocking *threshold* reads, i.e., `get`

```
let par _ = put lv (_,4)
        _ = put lv (3,_)
        x = getSnd lv
in x
```

- See intermediate state of the data structure

LVish

- A Haskell library for programming with LVars
- POPL '14, PLDI '14

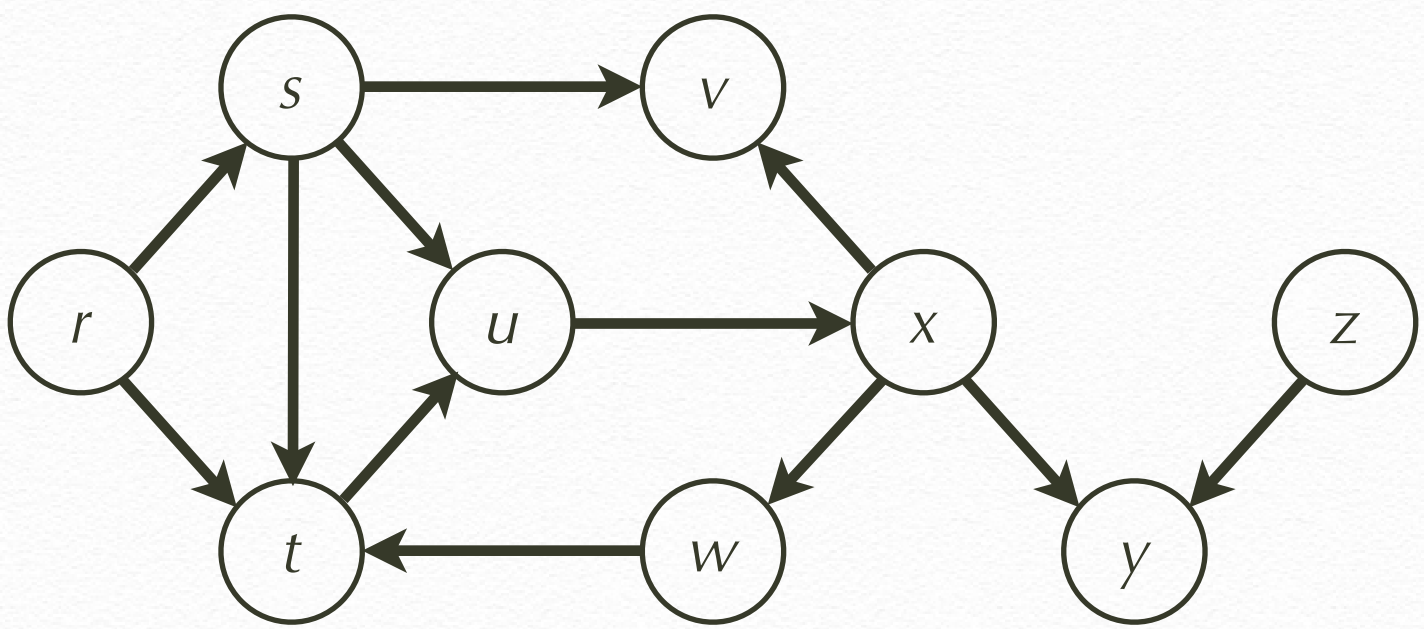
Handlers in LVish

- Callback functions
- Added to an LVar (using `addHandler`)
- Spawned every time an LVar *changes*

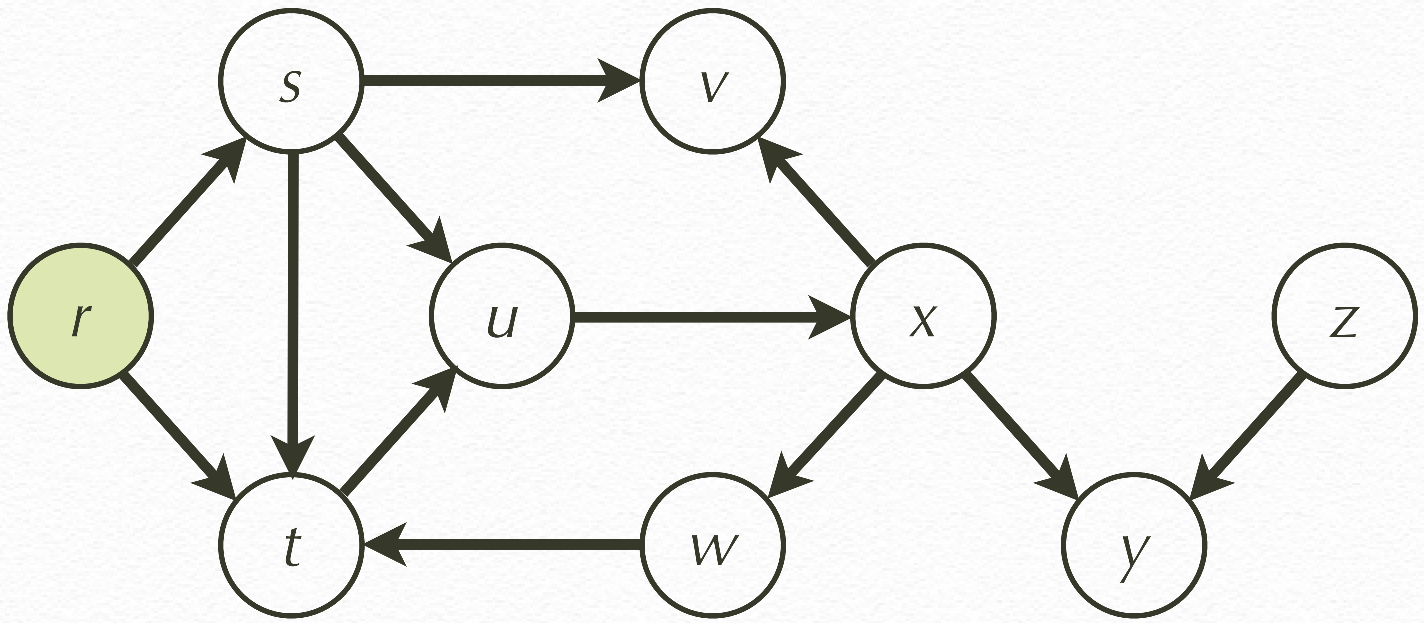
Connected component

```
connected :: Graph → NodeID → Par (ISet NodeID)
connected g startV = do
   seen ← newEmptySet
   addHandler seen
      (λ nd → parMapM (putInSet seen) (nbrs g nd))
   putInSet seen startV
   return seen
```

Connected component

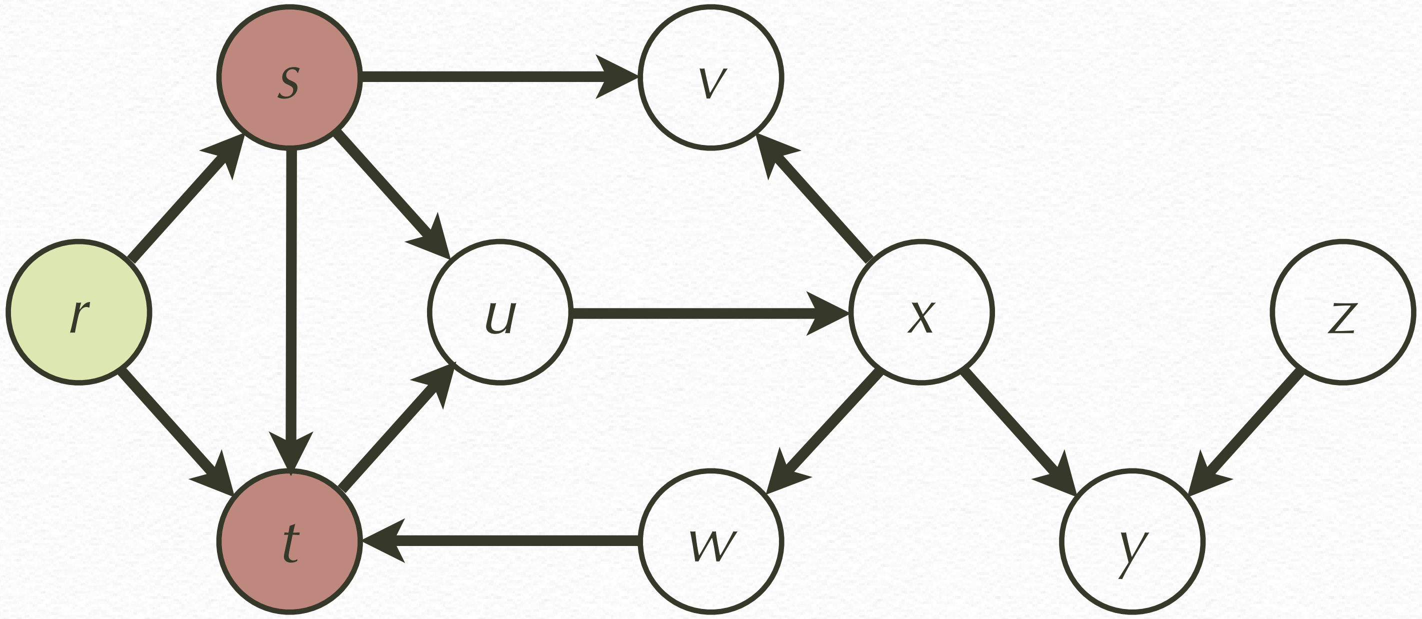


Connected component

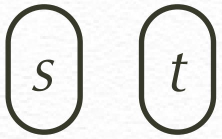



```
putInSet seen startV
```

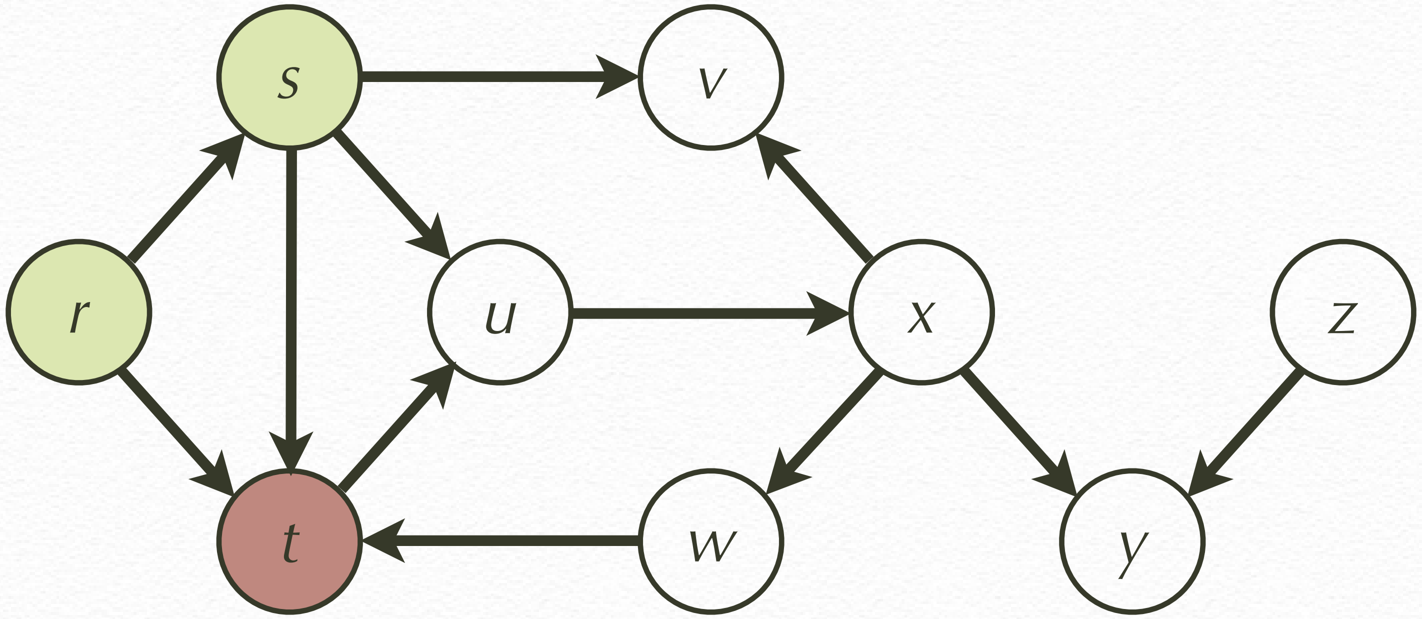
Connected component



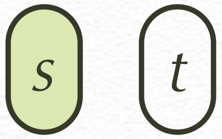

```
λ nd → parMapM (putInSet seen) (nbrs g nd)
```



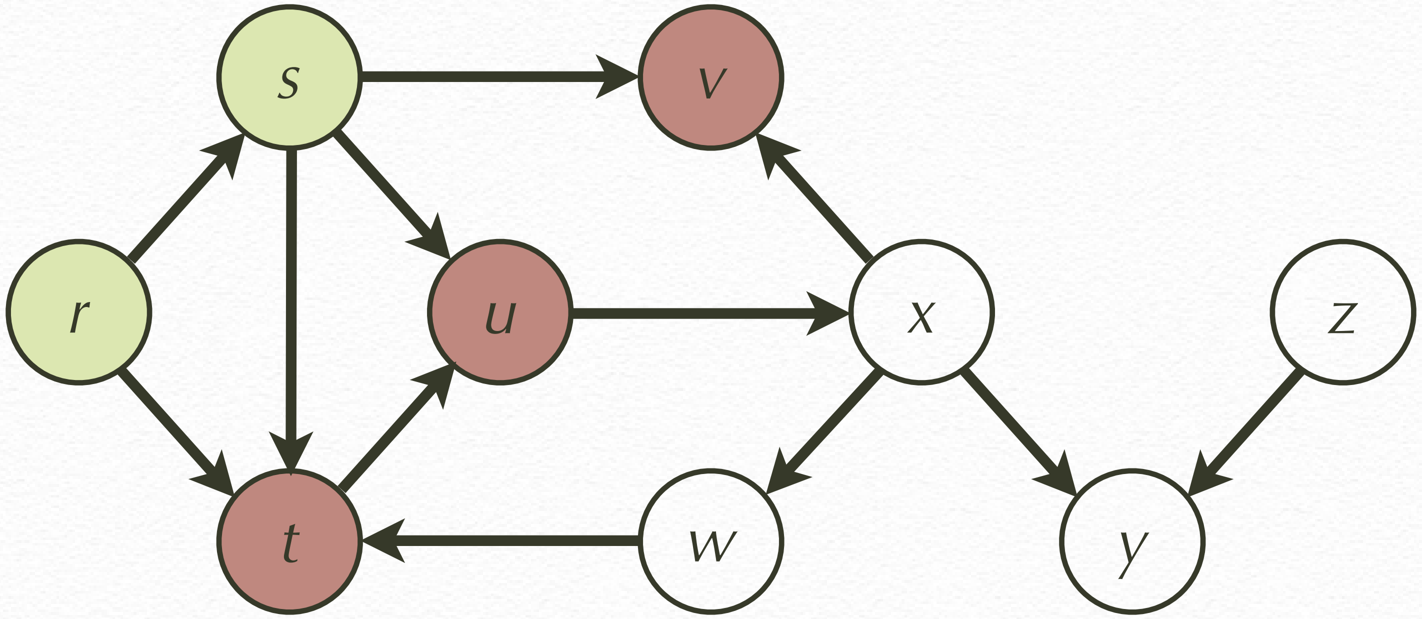
Connected component



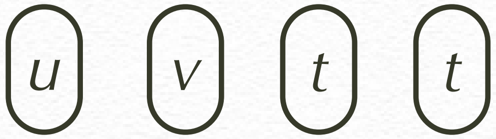

```
λ nd → parMapM (putInSet seen) (nbrs g nd)
```



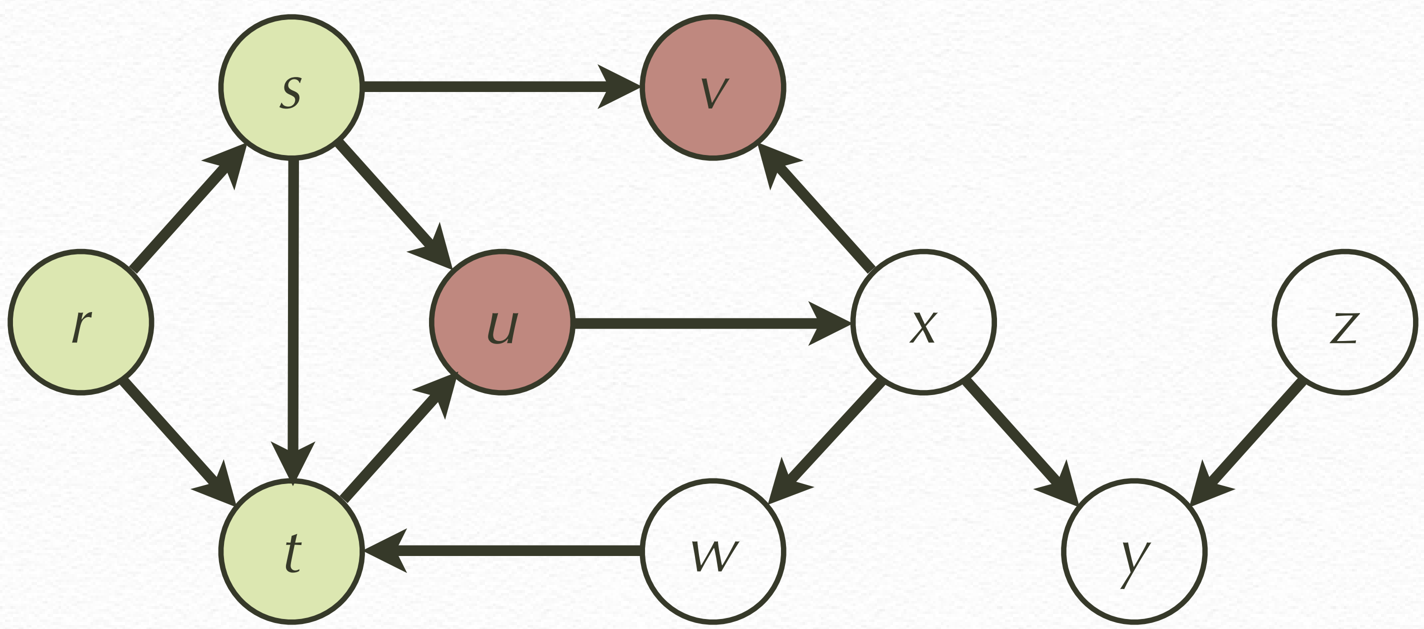
Connected component



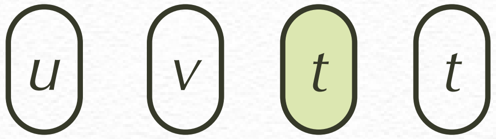
`λ nd → parMapM (putInSet seen) (nbrs g nd)`



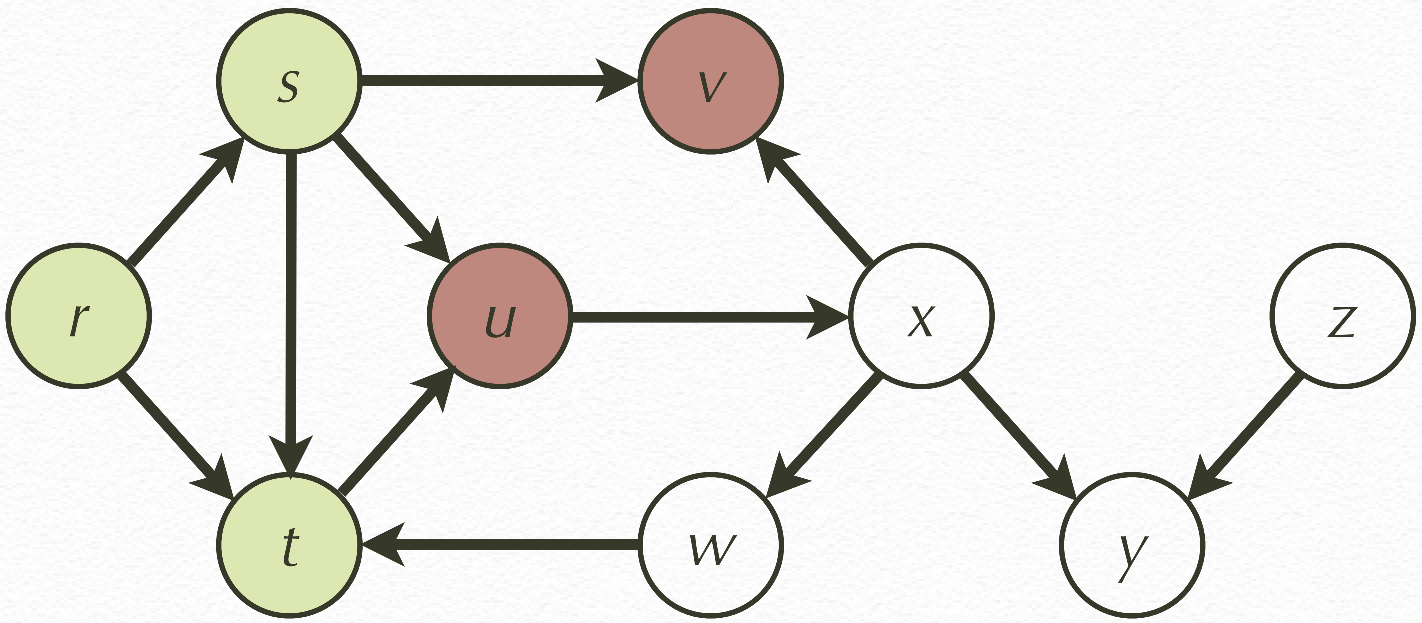
Connected component



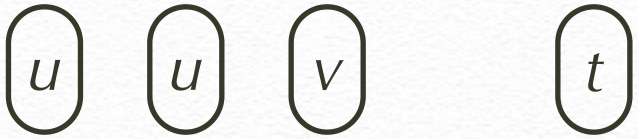

```
λ nd → parMapM (putInSet seen) (nbrs g nd)
```



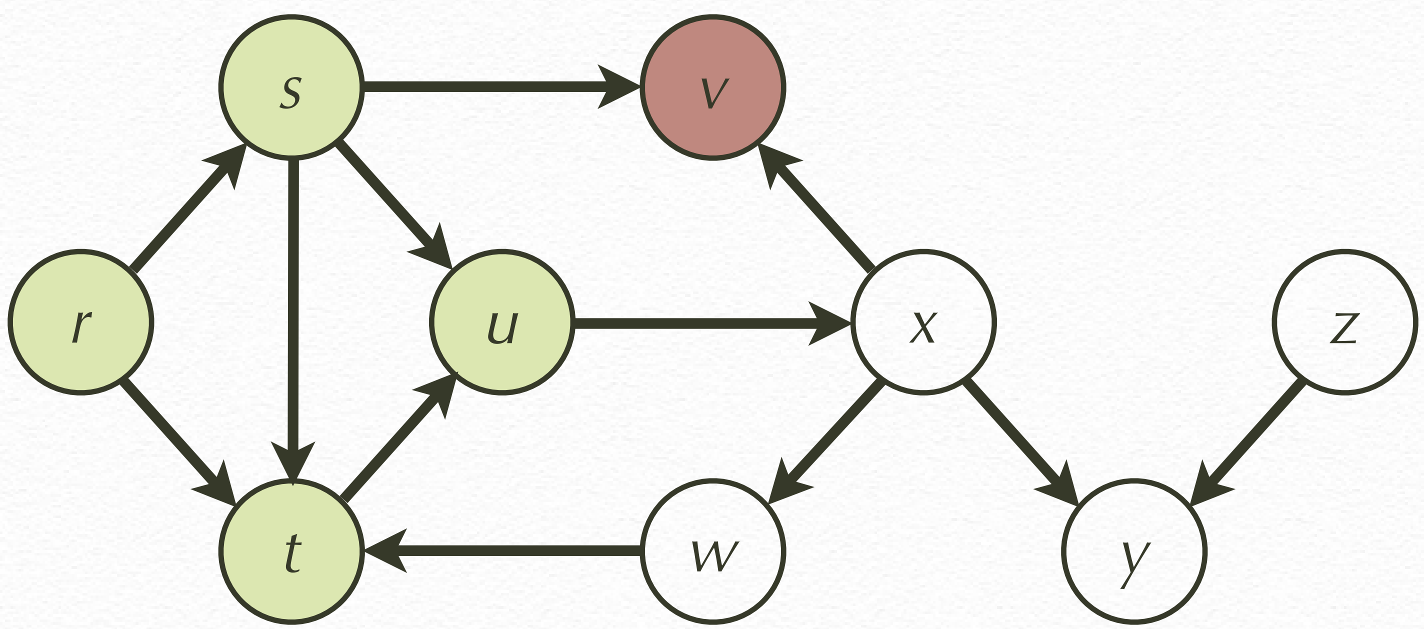
Connected component



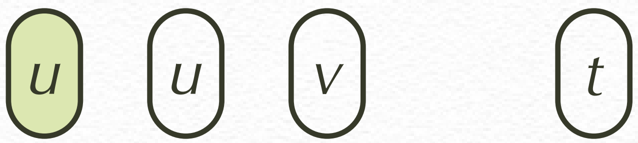

```
λ nd → parMapM (putInSet seen) (nbrs g nd)
```



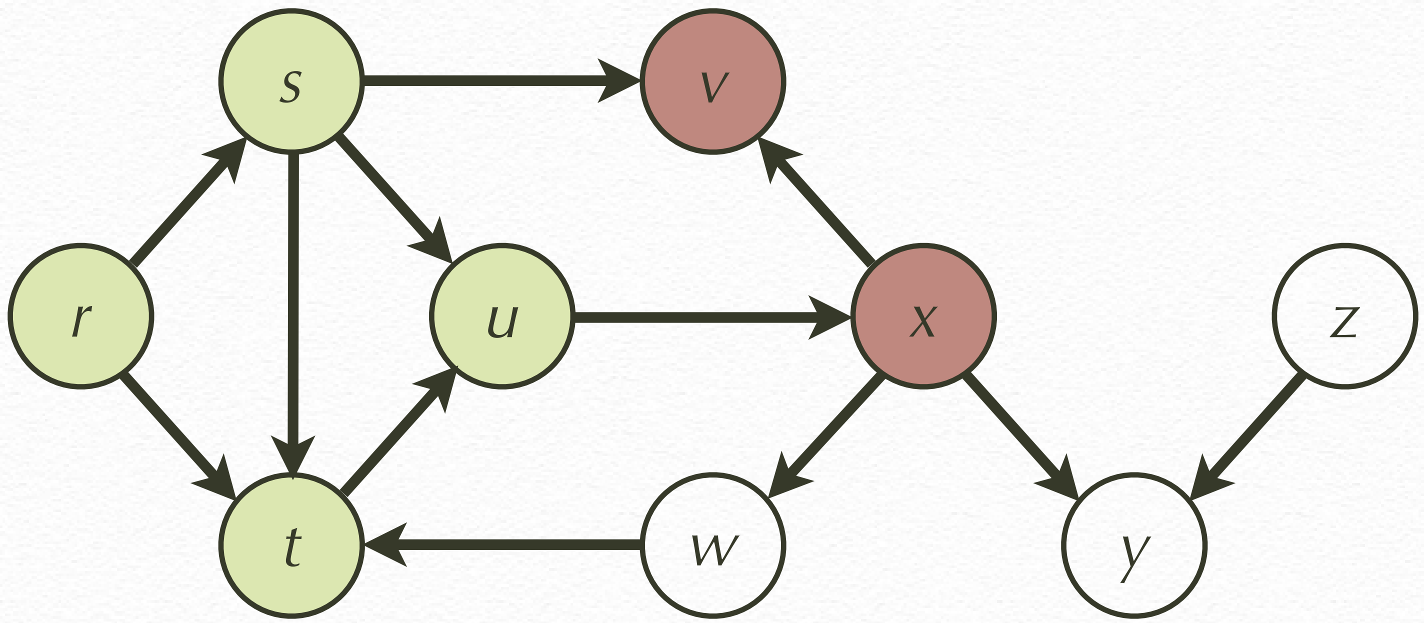
Connected component



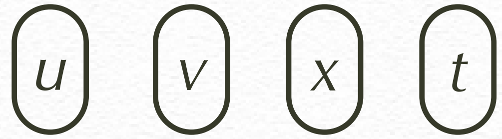

```
λ nd → parMapM (putInSet seen) (nbrs g nd)
```



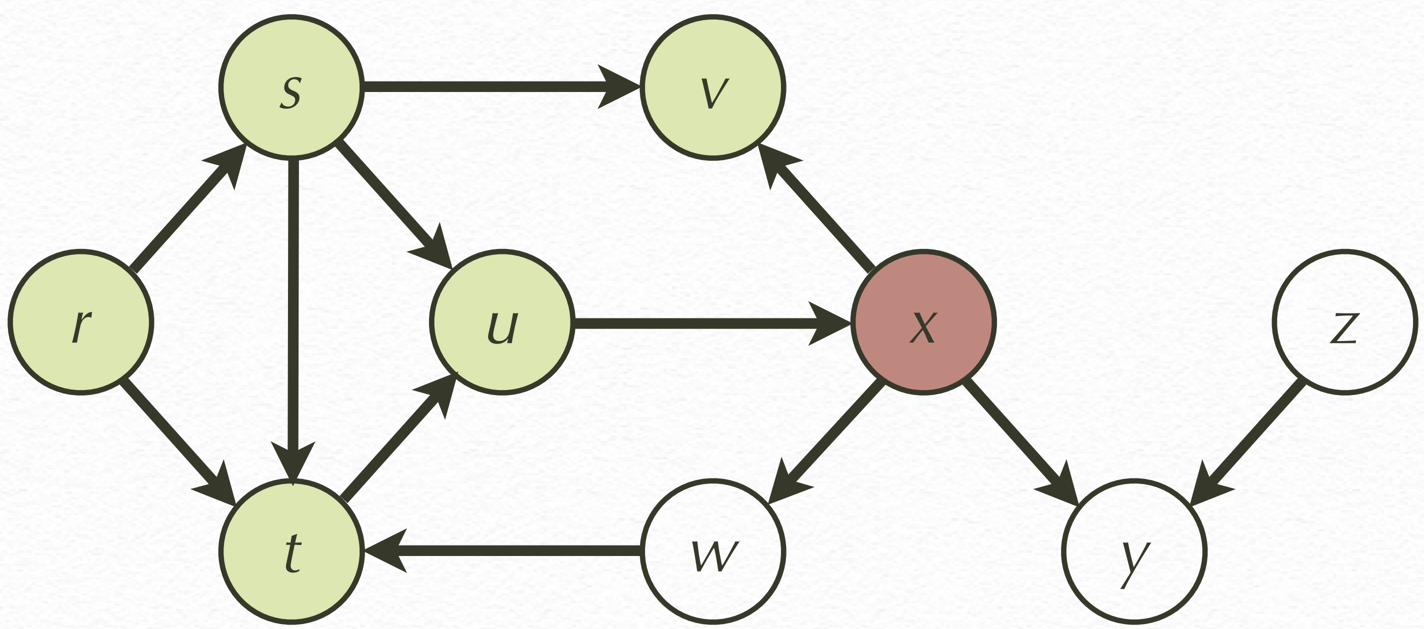
Connected component



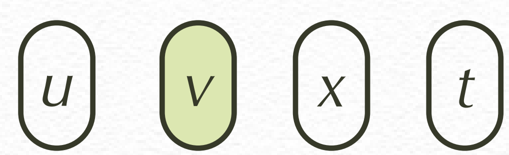

```
λ nd → parMapM (putInSet seen) (nbrs g nd)
```



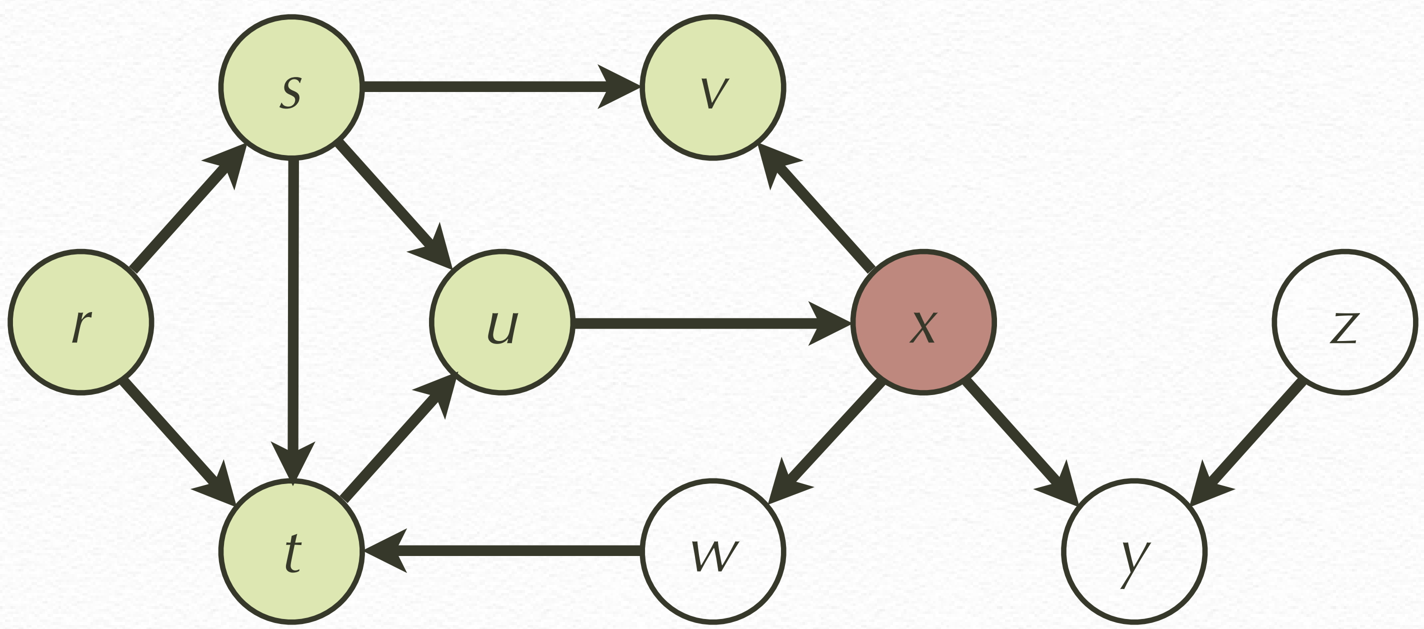
Connected component



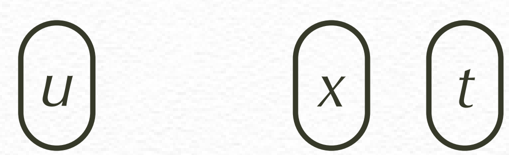

```
λ nd → parMapM (putInSet seen) (nbrs g nd)
```



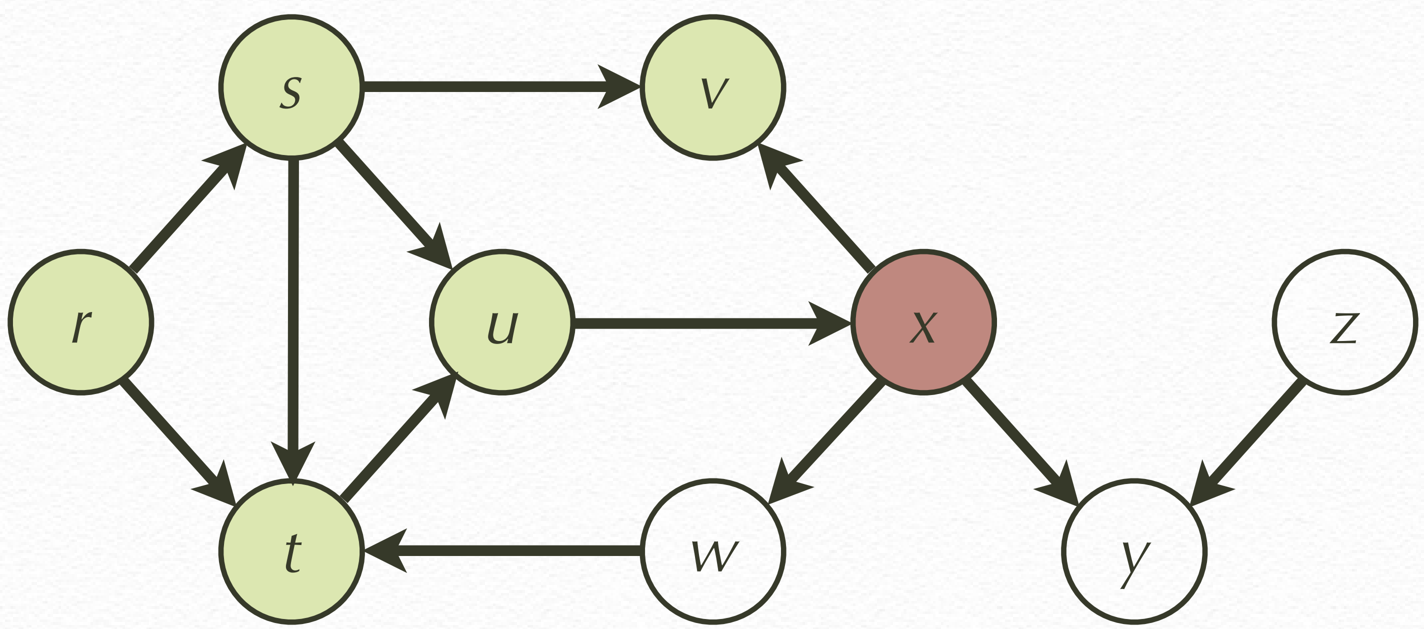
Connected component



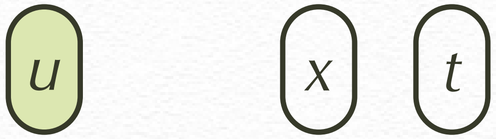

```
λ nd → parMapM (putInSet seen) (nbrs g nd)
```



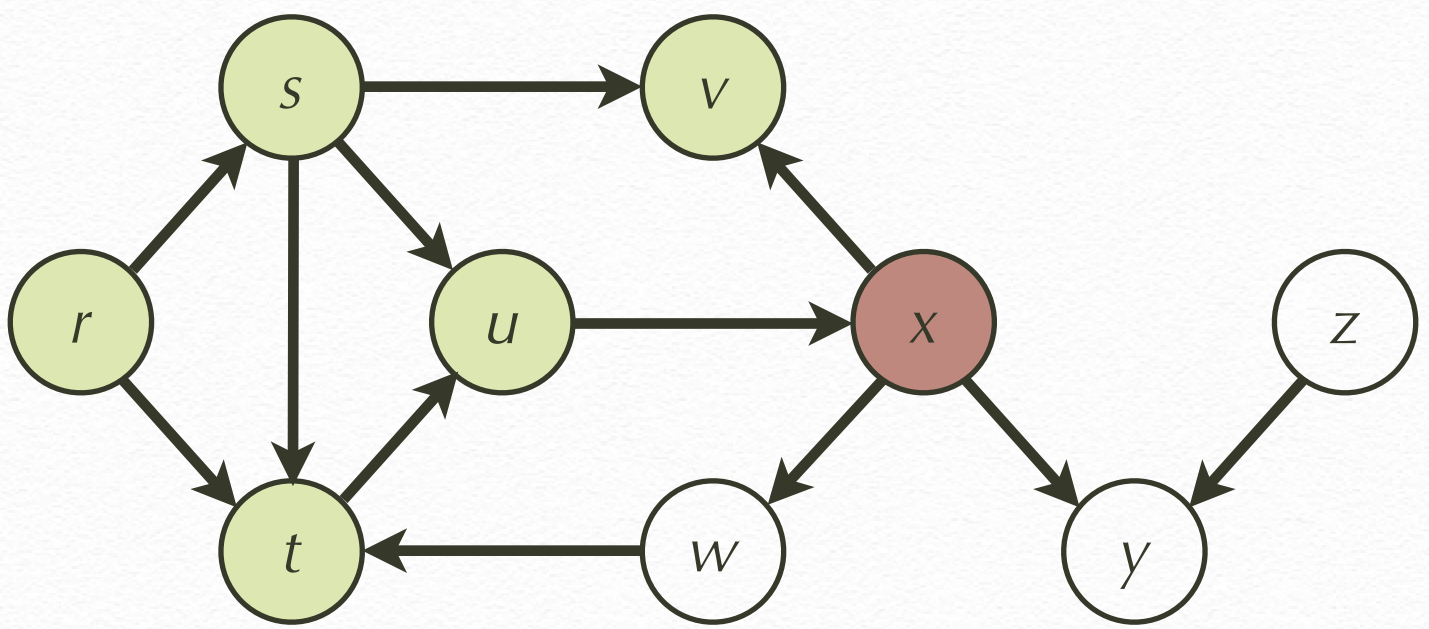
Connected component



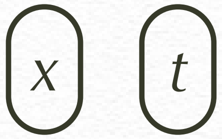

```
λ nd → parMapM (putInSet seen) (nbrs g nd)
```



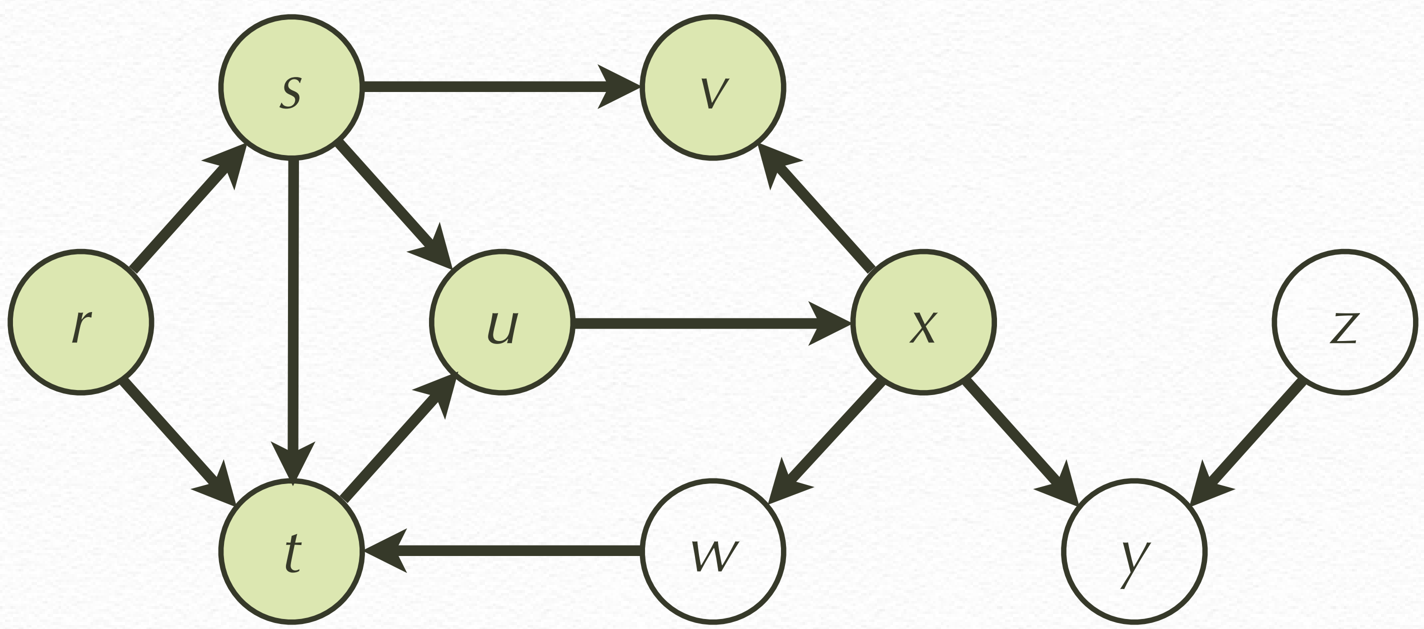
Connected component



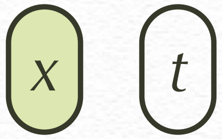

```
λ nd → parMapM (putInSet seen) (nbrs g nd)
```



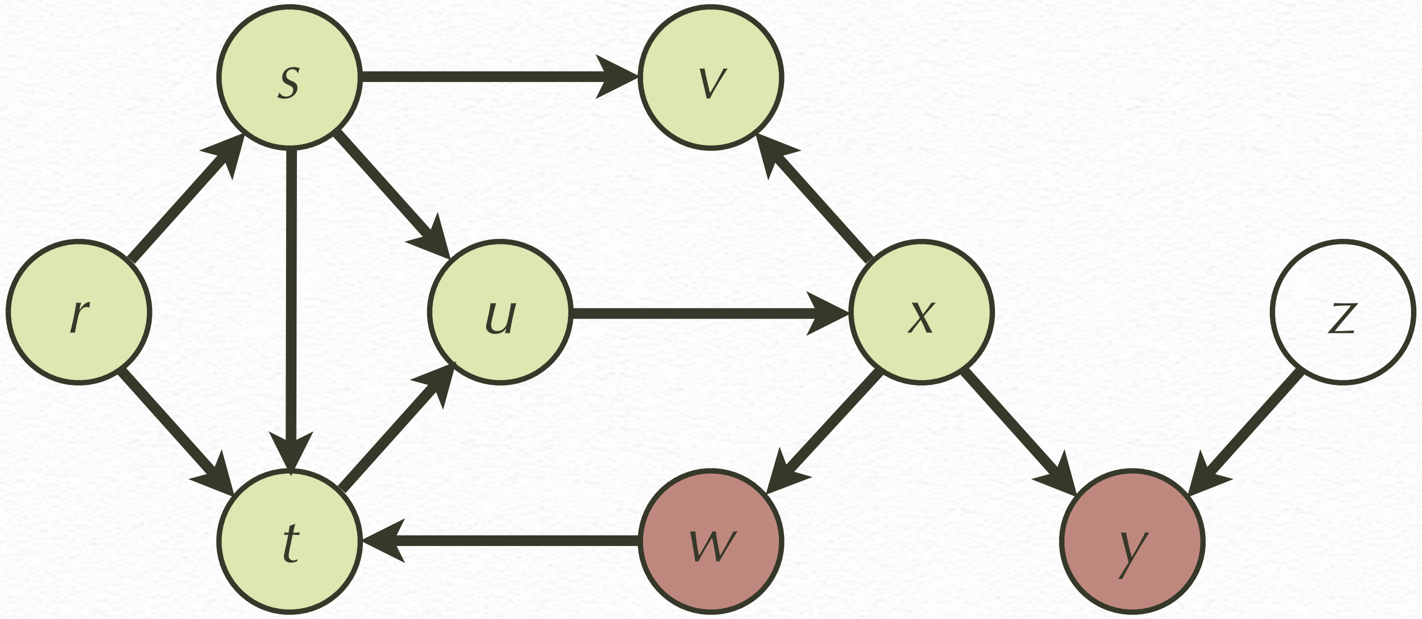
Connected component



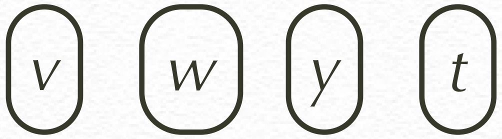

```
λ nd → parMapM (putInSet seen) (nbrs g nd)
```



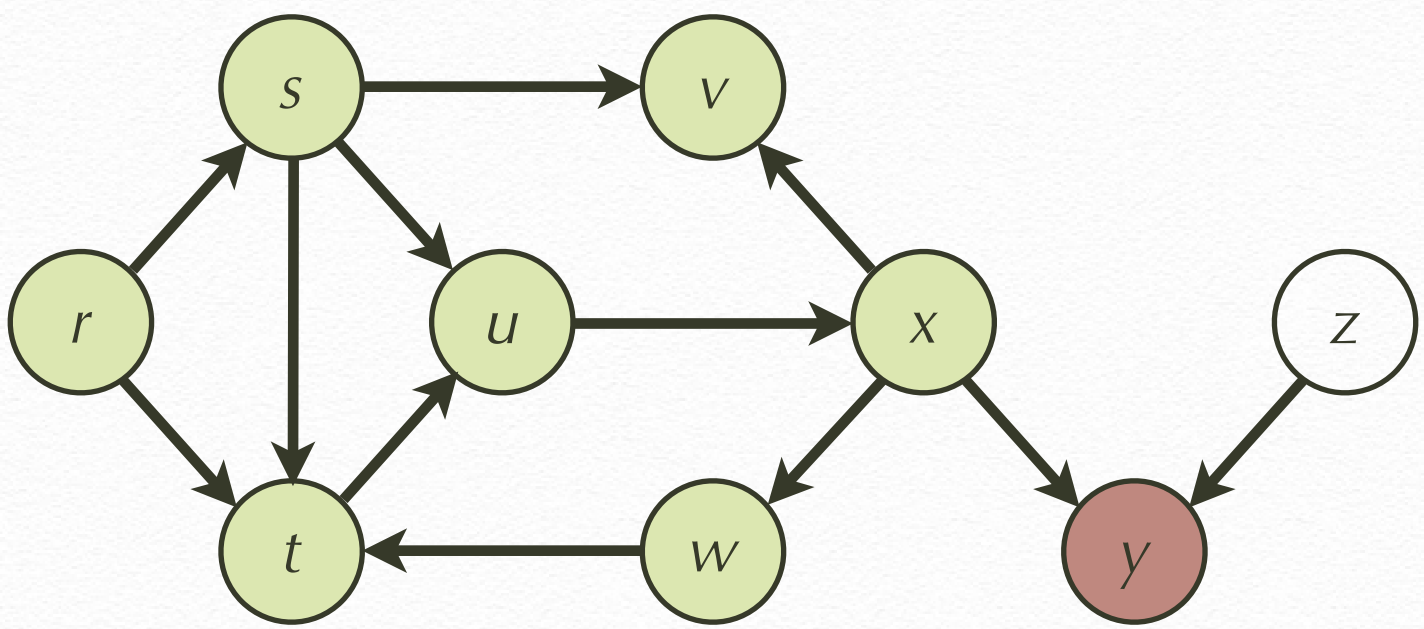
Connected component



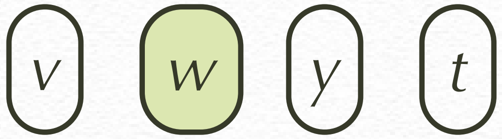

```
λ nd → parMapM (putInSet seen) (nbrs g nd)
```



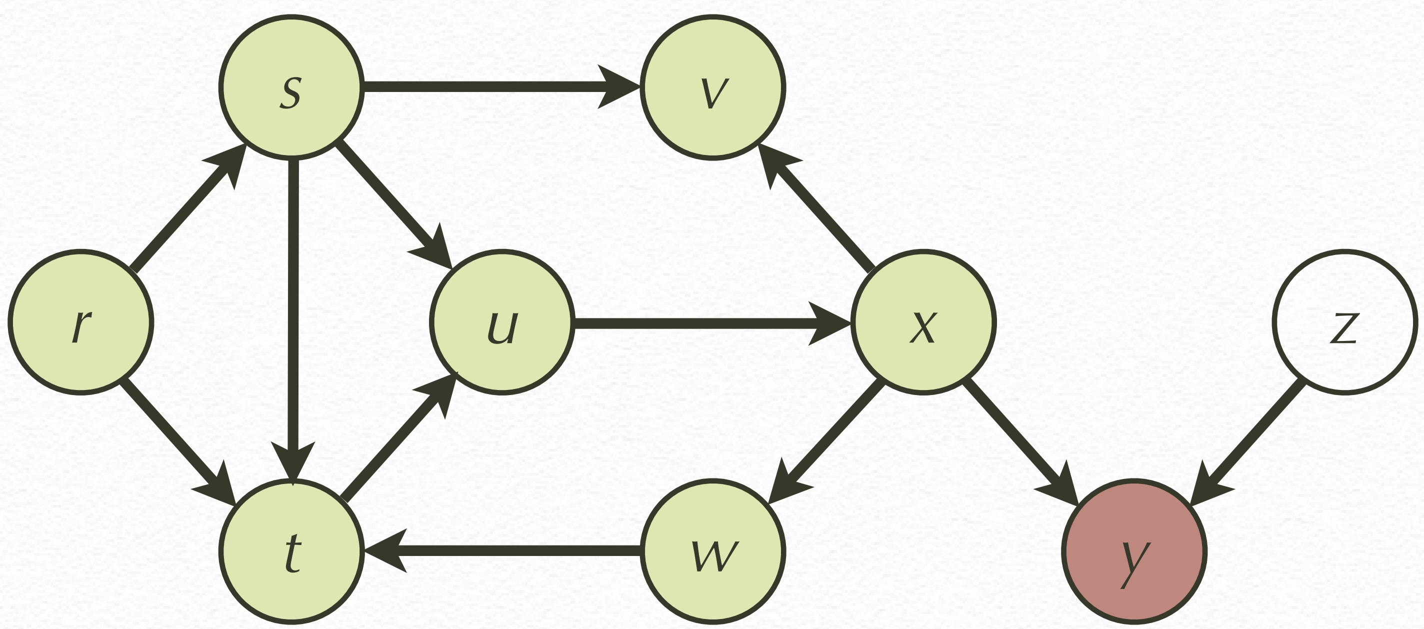
Connected component



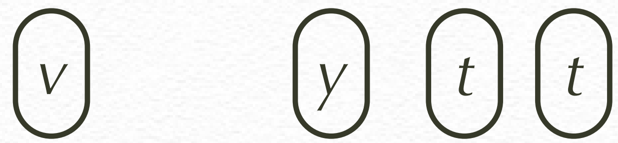

```
λ nd → parMapM (putInSet seen) (nbrs g nd)
```



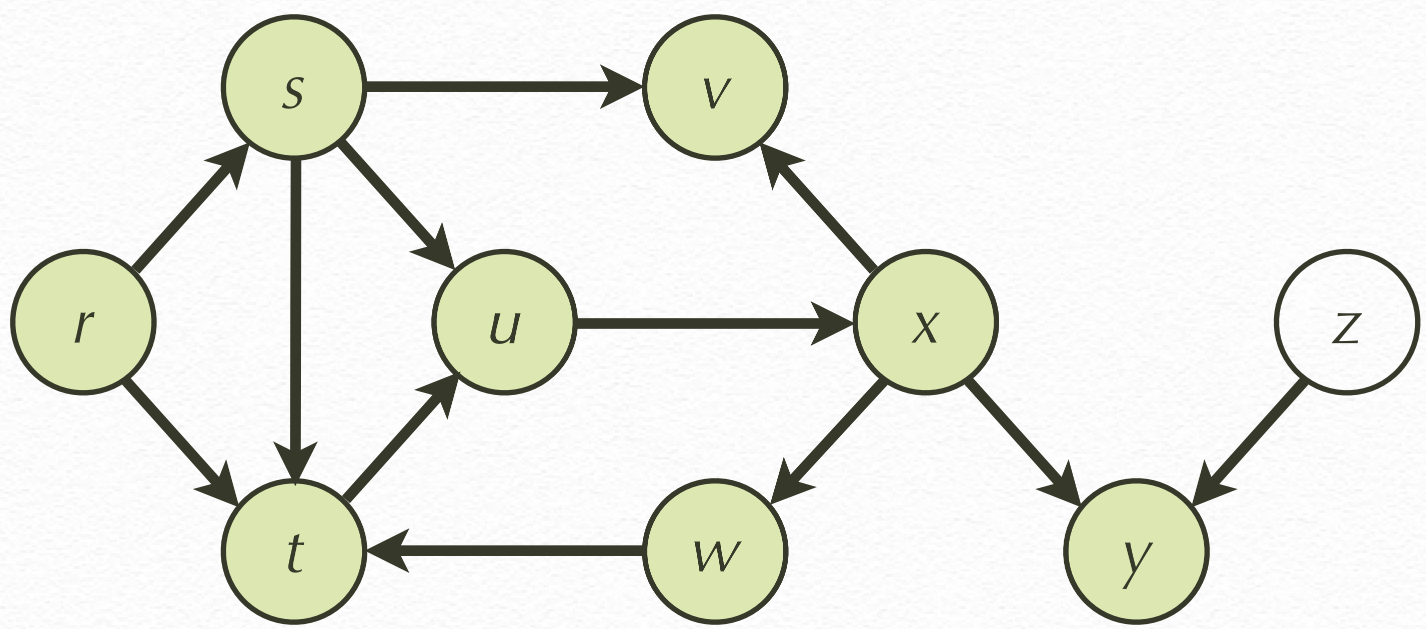
Connected component



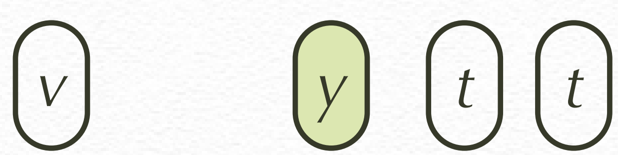

```
λ nd → parMapM (putInSet seen) (nbrs g nd)
```



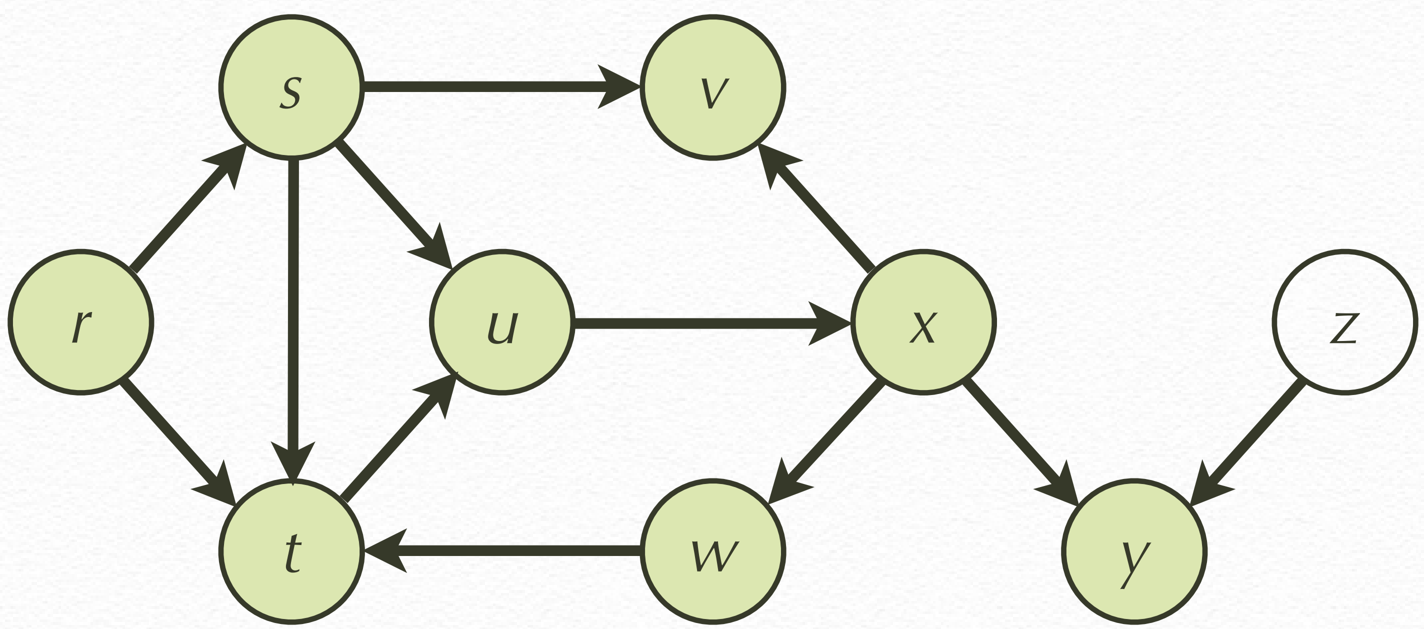
Connected component




```
λ nd → parMapM (putInSet seen) (nbrs g nd)
```

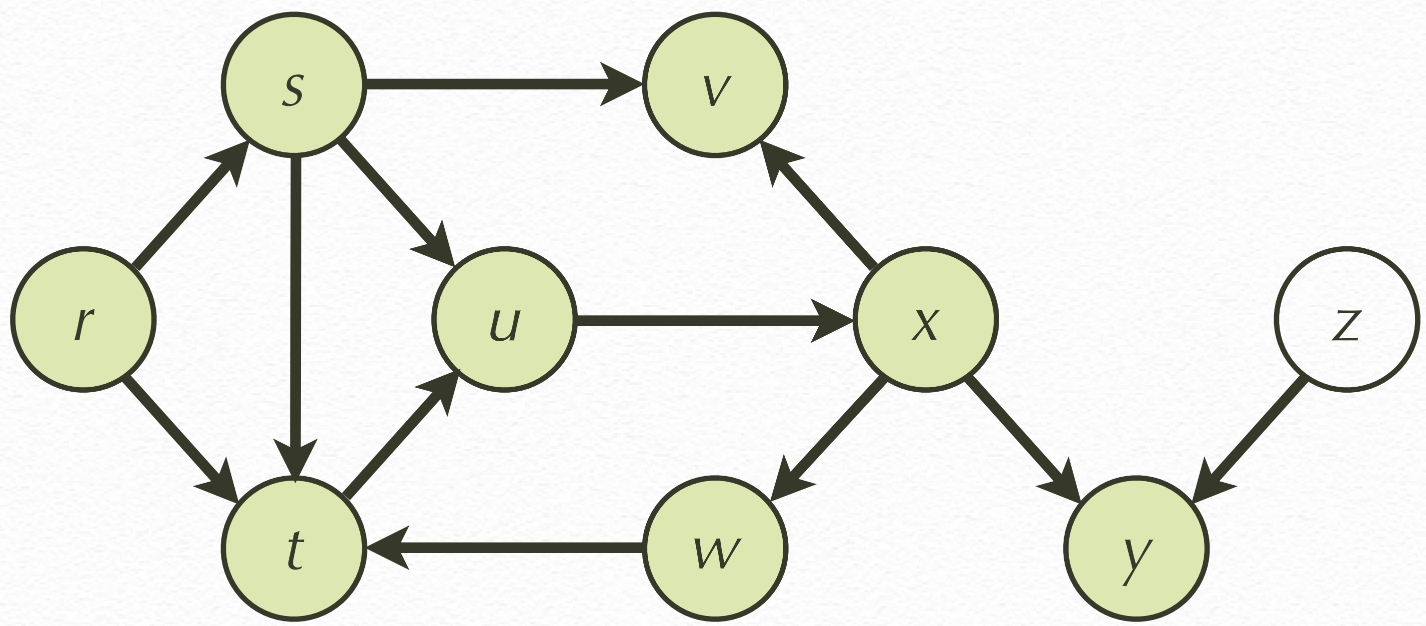


Connected component



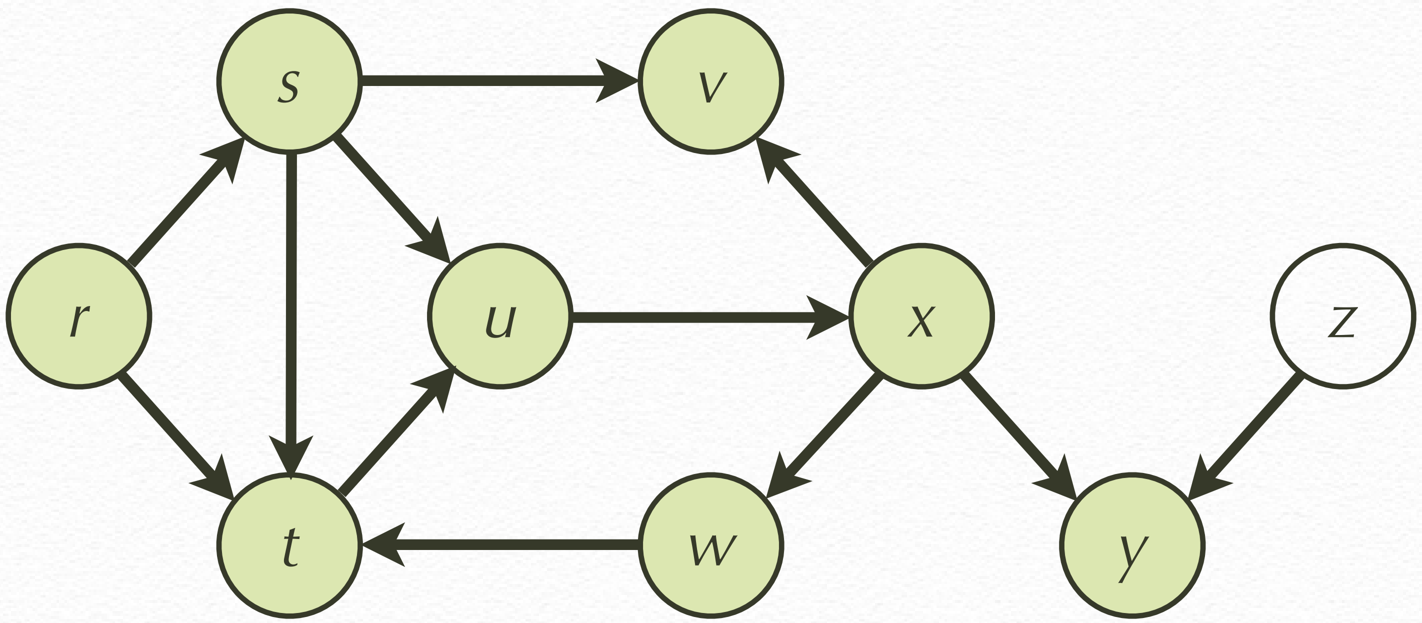

```
λ nd → parMapM (putInSet seen) (nbrs g nd)
```

Connected component



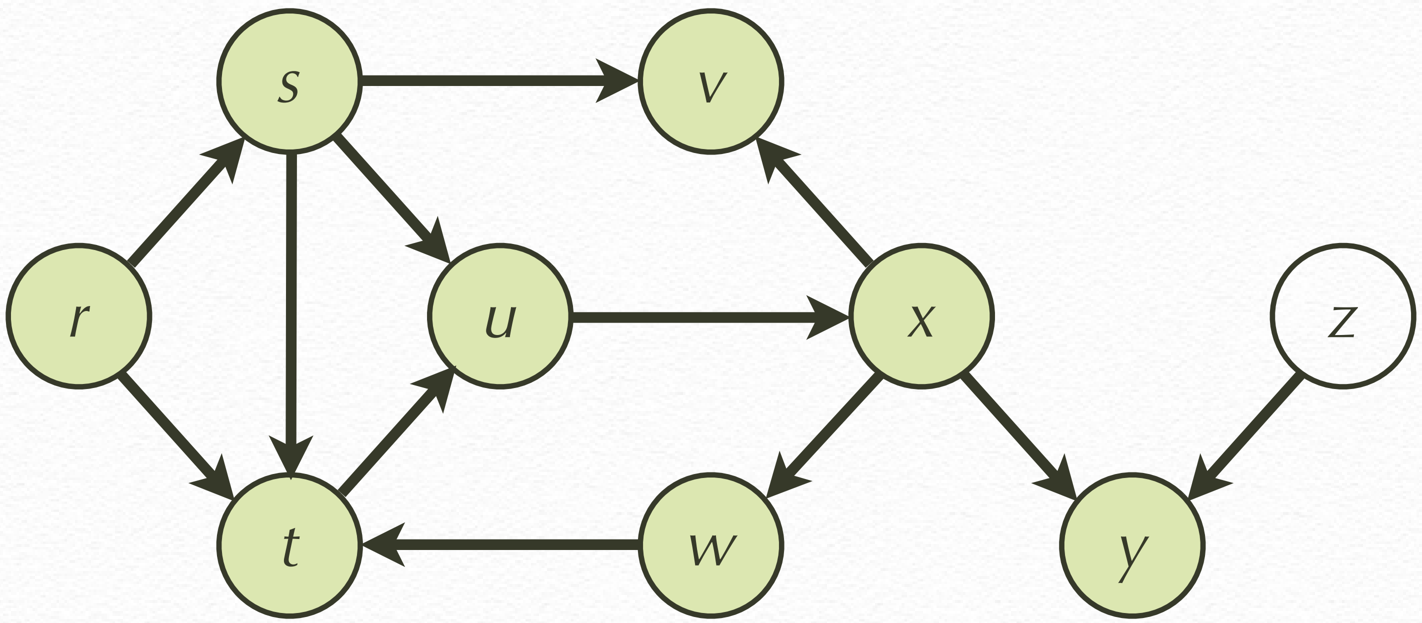

```
λ nd → parMapM (putInSet seen) (nbrs g nd)
```

Connected component



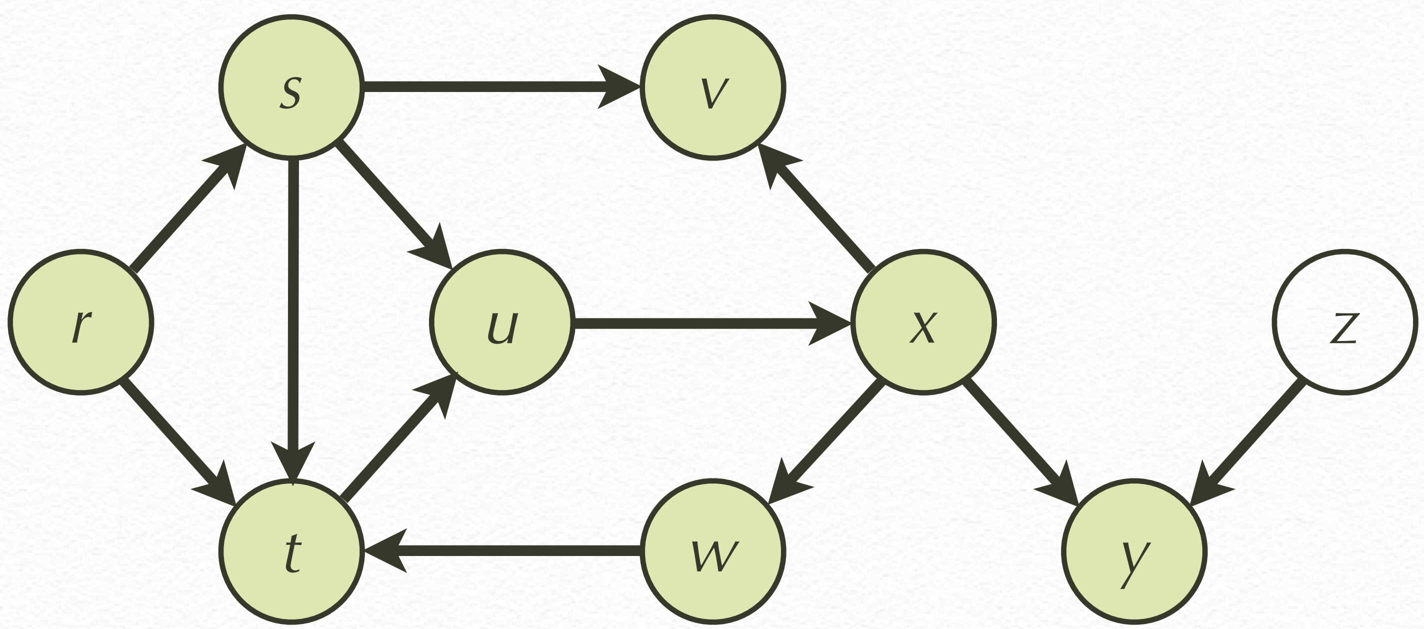

```
λ nd → parMapM (putInSet seen) (nbrs g nd)
```

Connected component

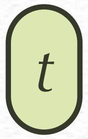



```
λ nd → parMapM (putInSet seen) (nbrs g nd)
```

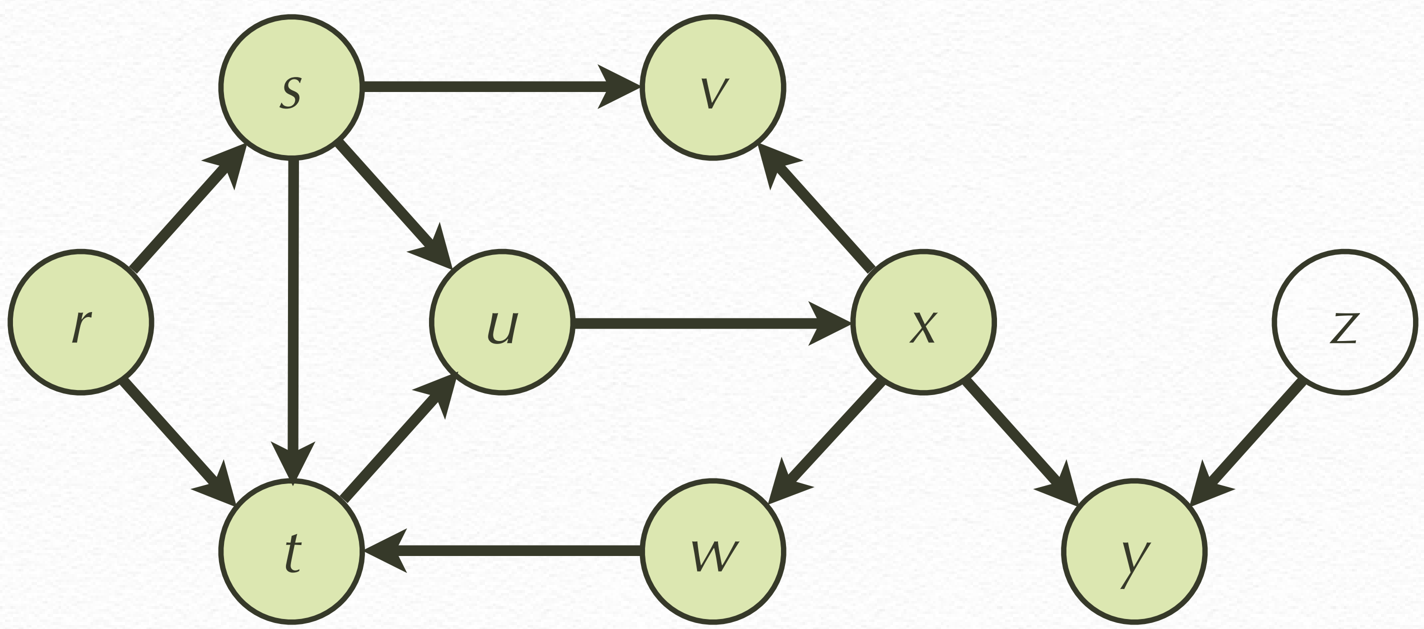
Connected component




```
λ nd → parMapM (putInSet seen) (nbrs g nd)
```



Connected component



Breadth-first search

- Similar to `connected`
- Return BFS tree topology rather than “`seen`” set

Deterministic spanning tree

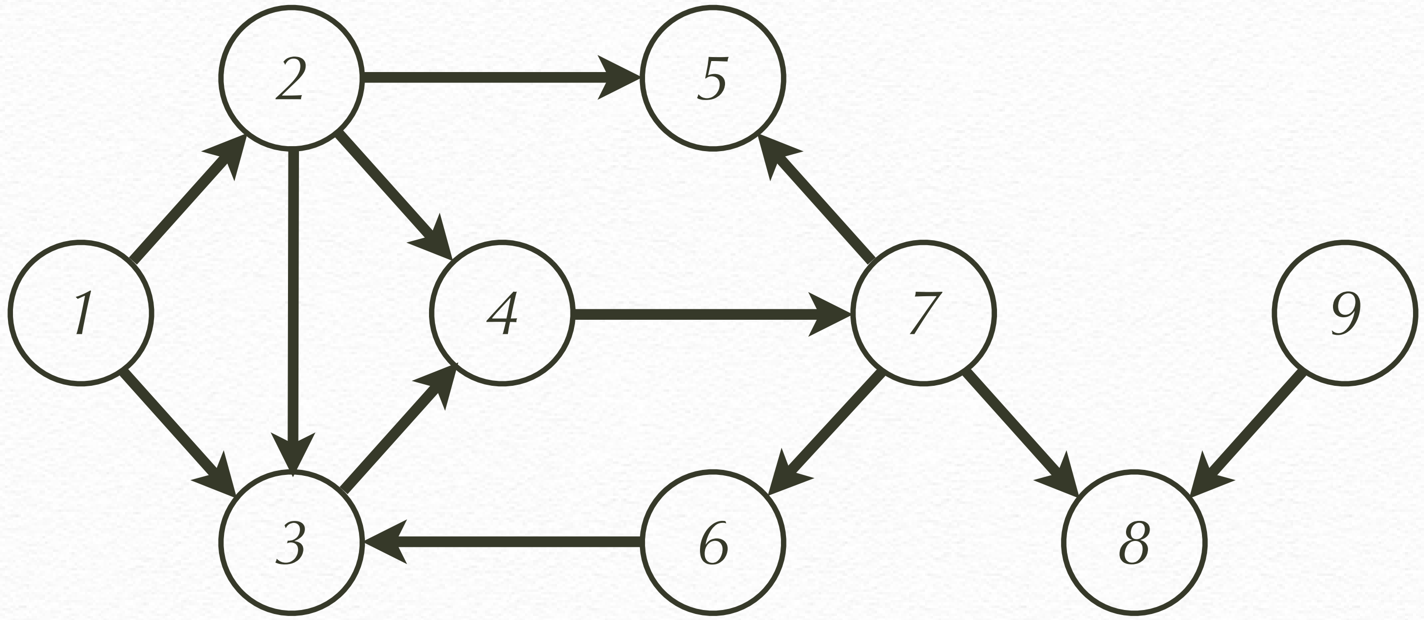
- Use a reduction variable to track highest priority parent
- *MinVar, a*n instance of LVars

Breadth-first search

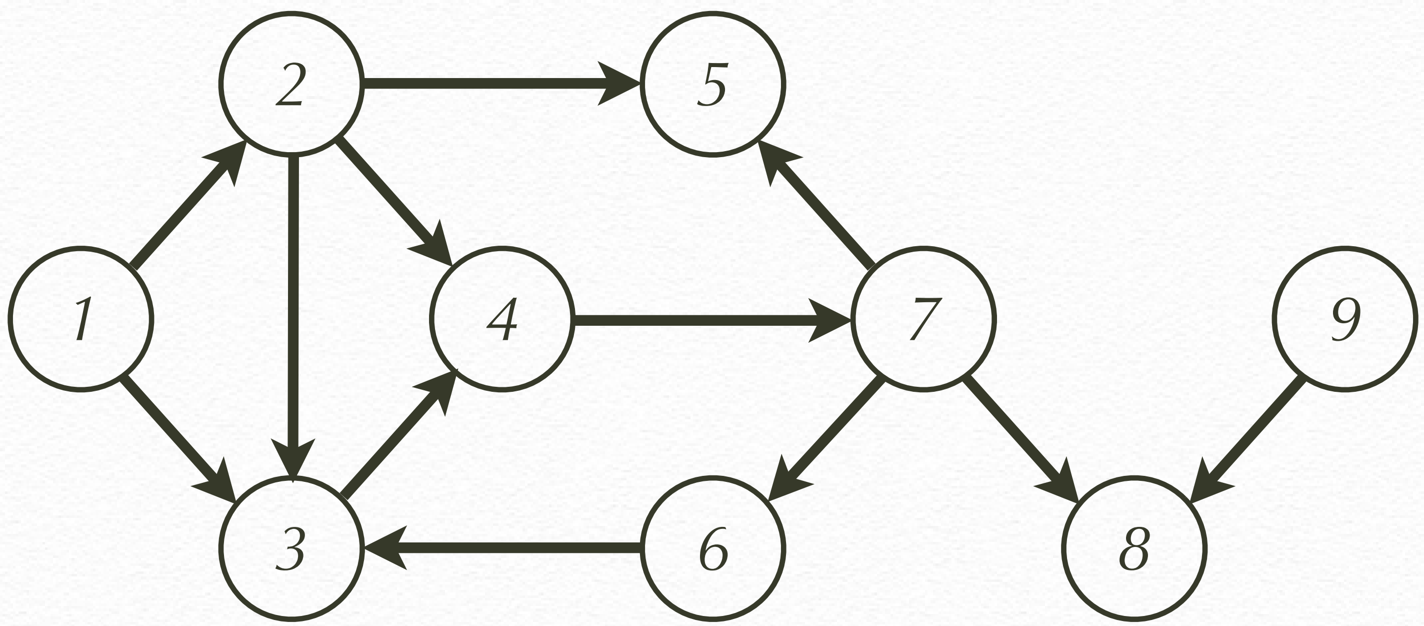
```haskell
import Data.LVar.MinVar
type NodeID = Int

bfsTree :: Graph → NodeID → Par (Vector MinVar)
bfsTree gr start = do
  let (Graph verts edges) = gr
  parents ← Vector.generateM (length verts)
                             (λ ix → newMinVar maxInt)
  seen ← newEmptySet
  let handler nd = mapM (eachNbr nd) (nbrs g nd)
      eachNbr nd nbr = do putInSet seen nbr
                          putMin parents nbr nd
  addHandler seen handler -- Register callback.
  putInSet seen start     -- Kick things off.
  return parents
```

Deterministic tree



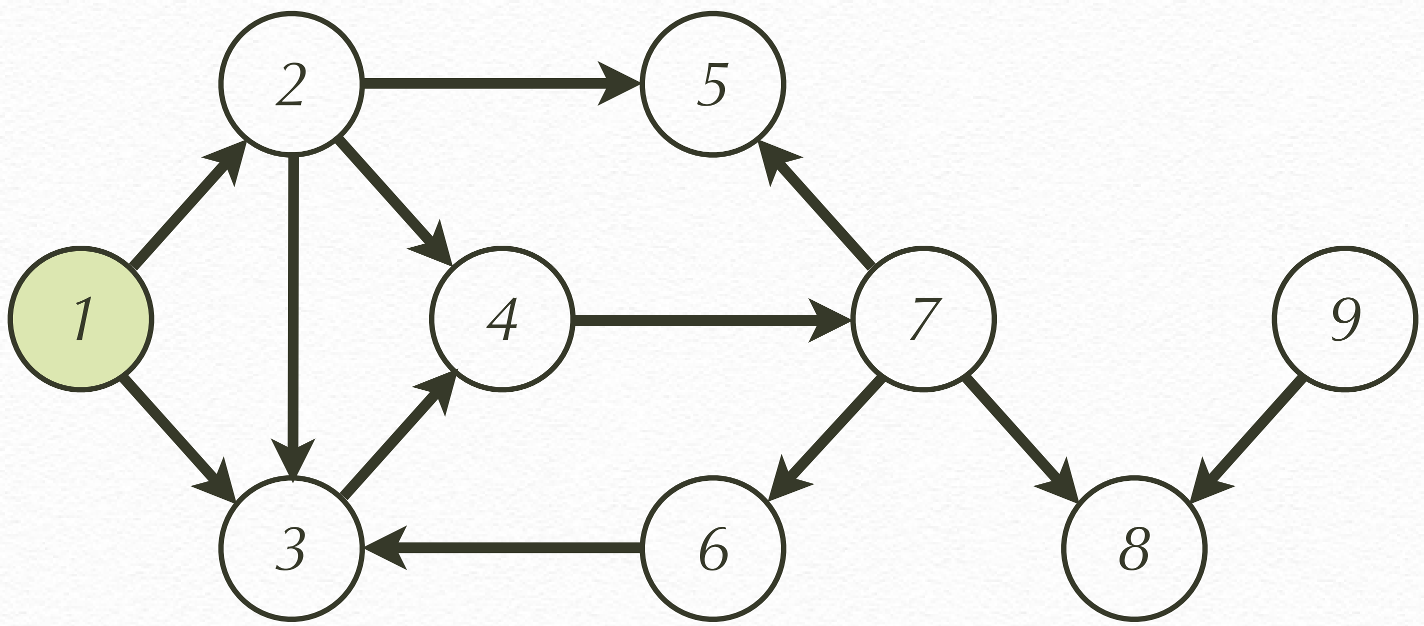
Deterministic tree




```
Vector.generateM (length verts) (λ ix → newMinVar maxInt)
```

maxInt	maxInt	maxInt	maxInt	maxInt	maxInt	maxInt	maxInt	maxInt
1	2	3	4	5	6	7	8	9

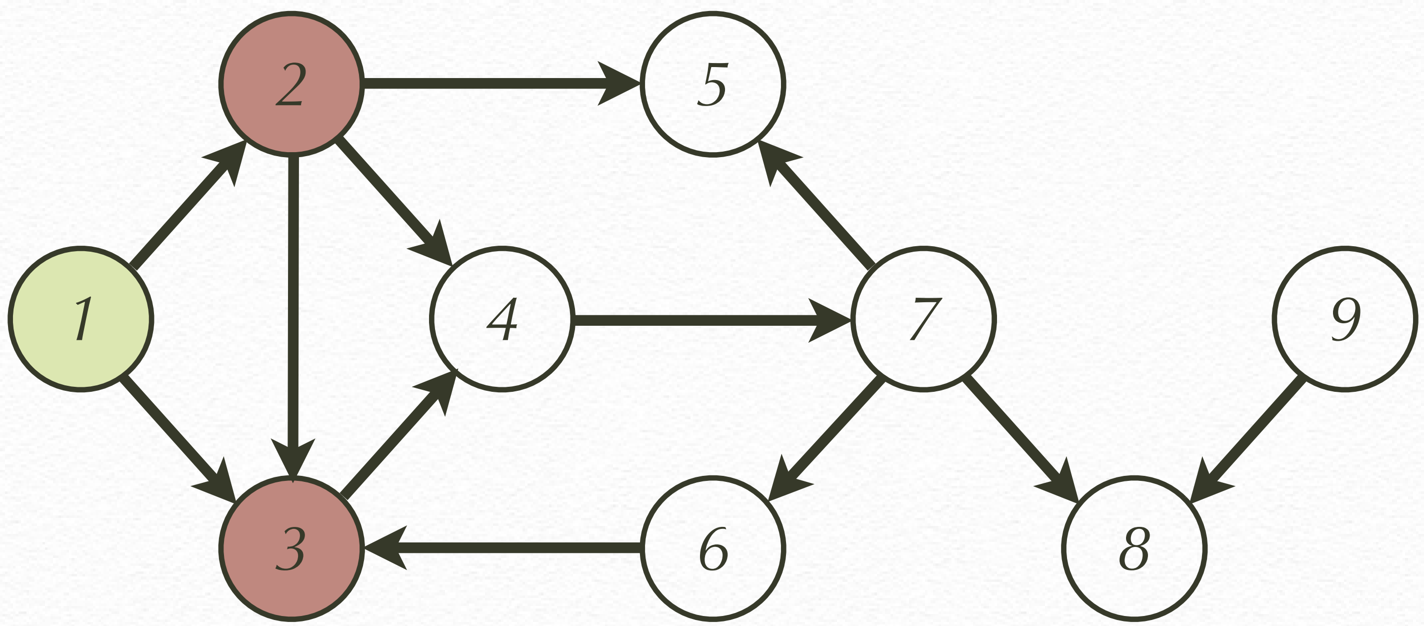
Deterministic tree



`putInSet seen start`

maxInt	maxInt	maxInt	maxInt	maxInt	maxInt	maxInt	maxInt	maxInt
1	2	3	4	5	6	7	8	9

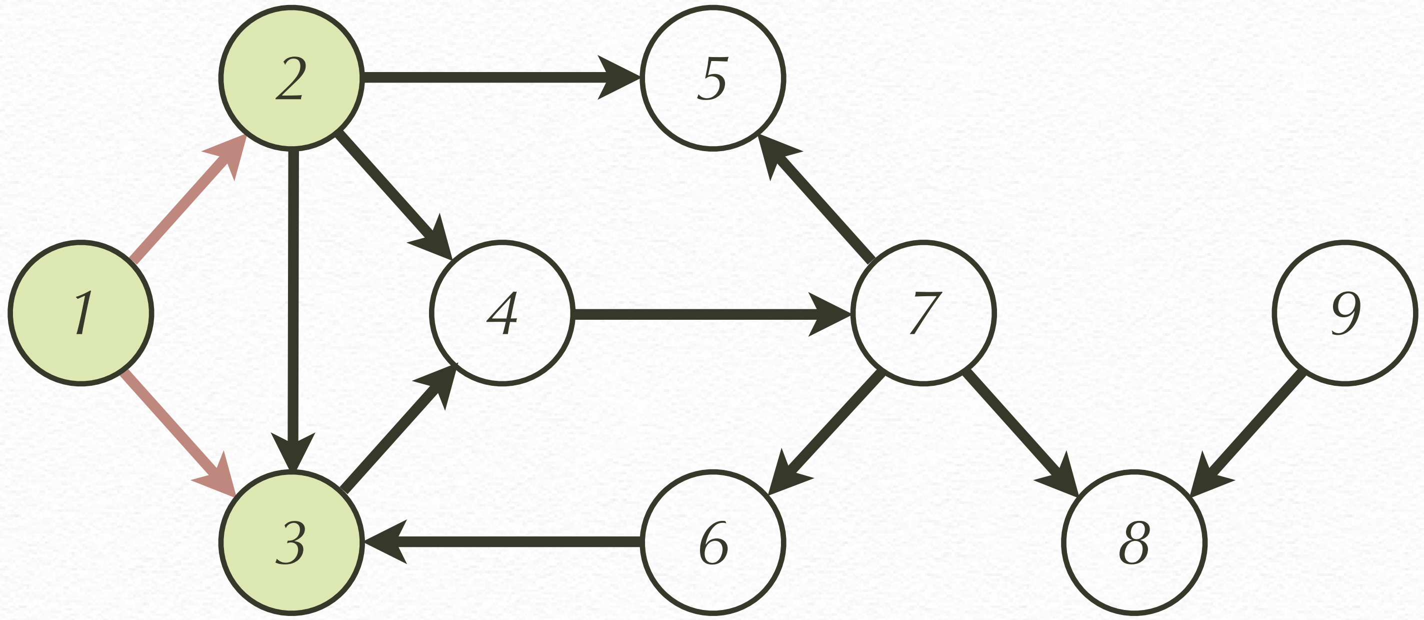
Deterministic tree




```
do putInSet seen nbr
   putMin parents nbr nd
```

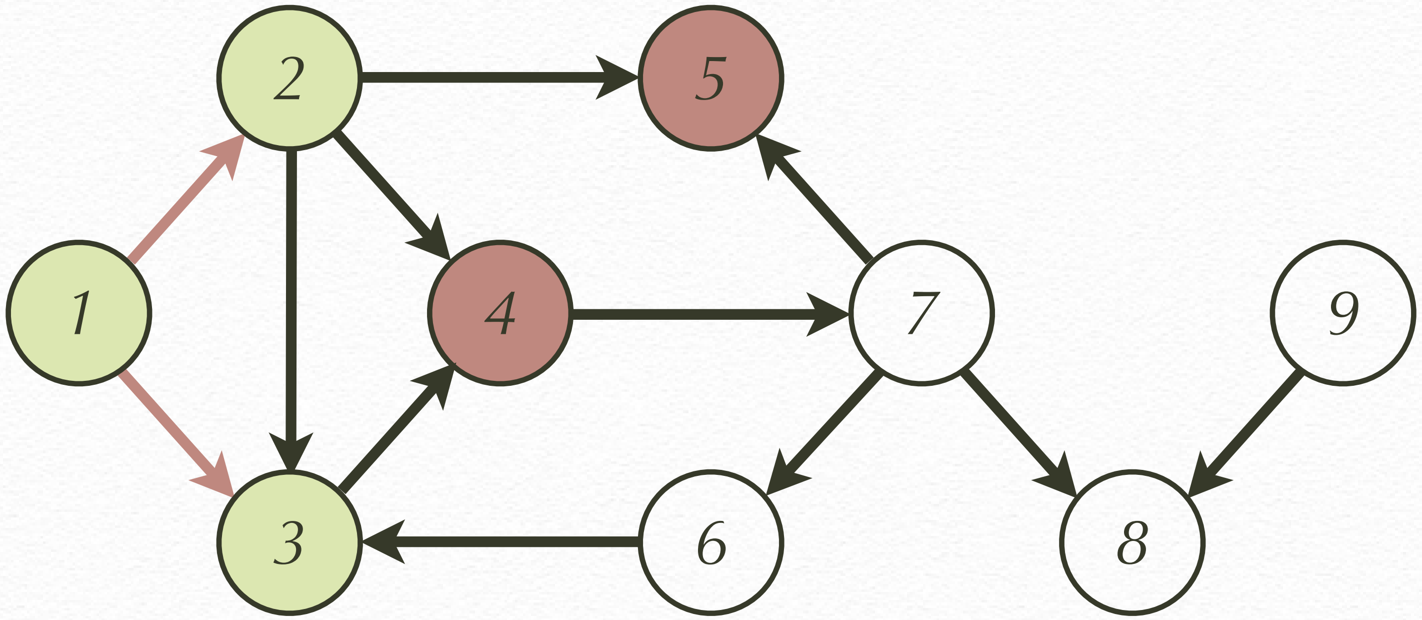
maxInt	maxInt	maxInt	maxInt	maxInt	maxInt	maxInt	maxInt	maxInt
1	2	3	4	5	6	7	8	9

Deterministic tree



maxInt	1	1	maxInt	maxInt	maxInt	maxInt	maxInt	maxInt
1	2	3	4	5	6	7	8	9

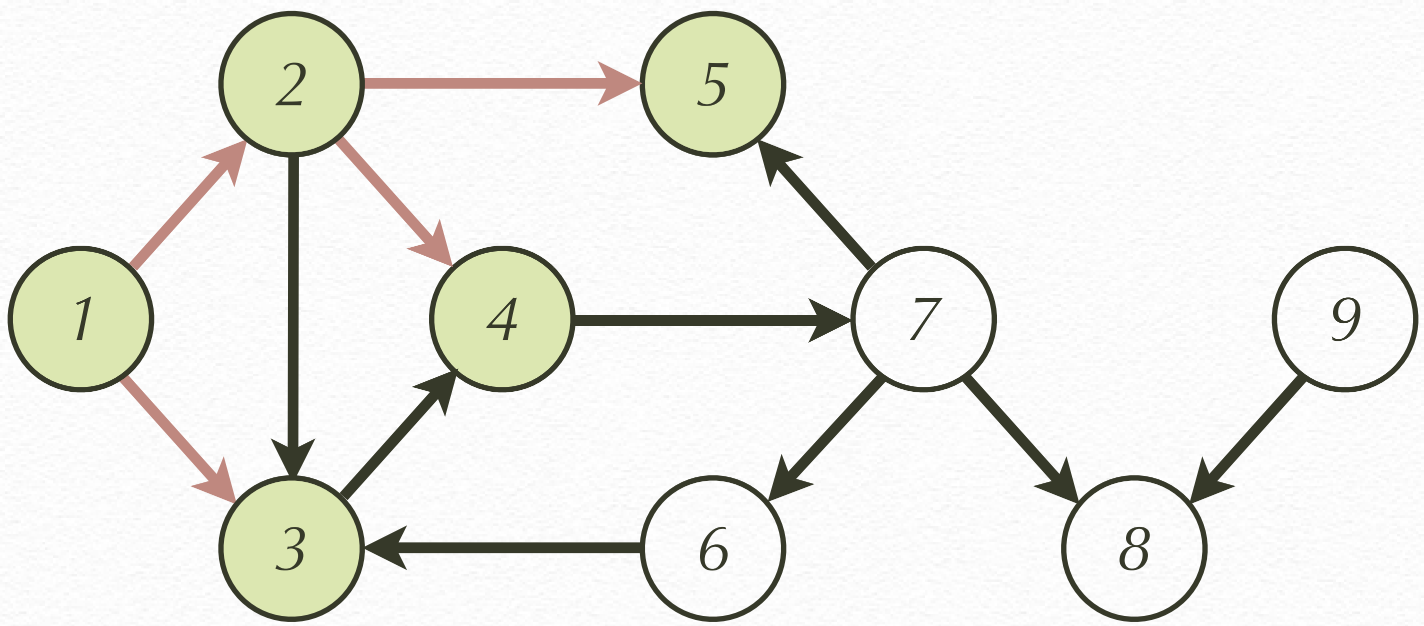
Deterministic tree




```
do putInSet seen nbr
   putMin parents nbr nd
```

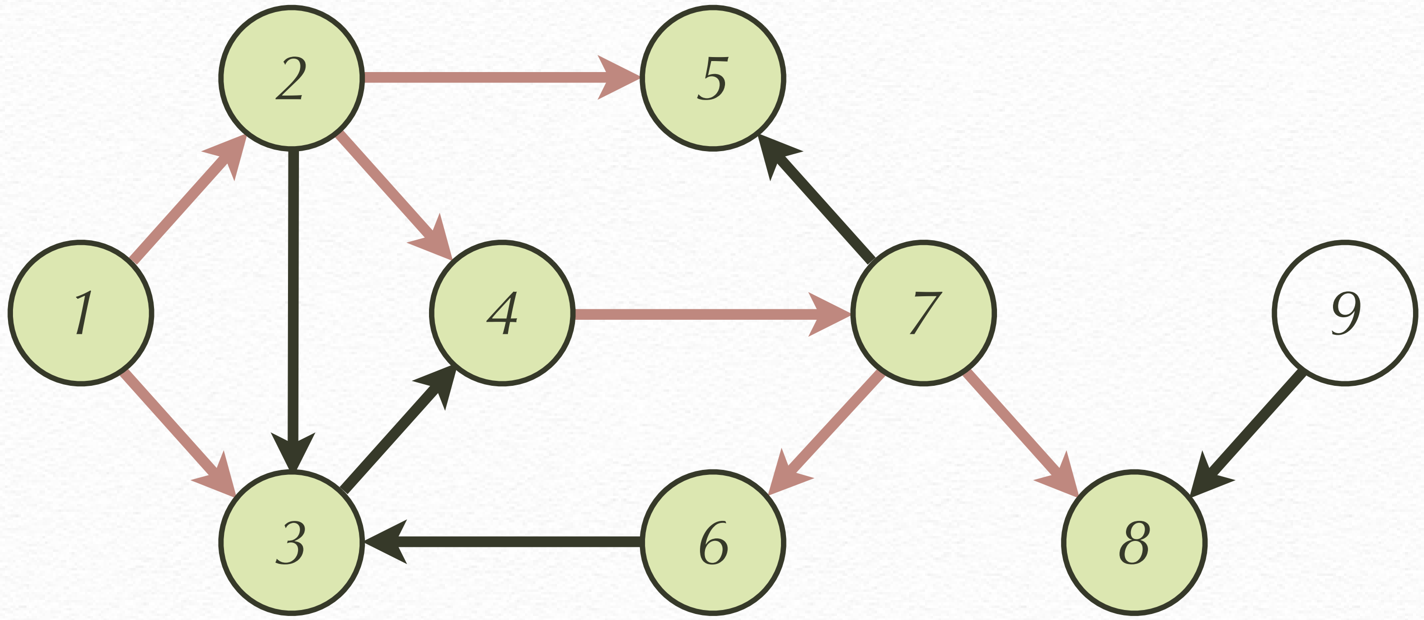
maxInt	1	1	maxInt	maxInt	maxInt	maxInt	maxInt	maxInt
1	2	3	4	5	6	7	8	9

Deterministic tree



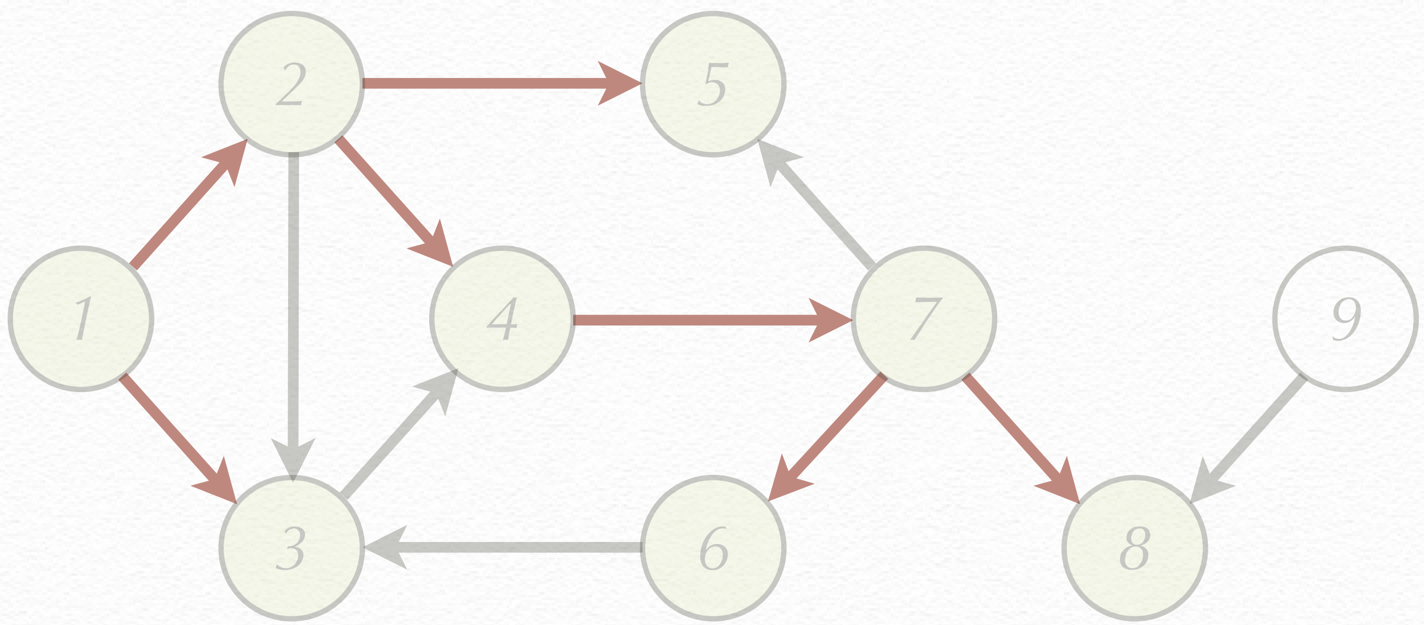
maxInt	1	1	2	2	maxInt	maxInt	maxInt	maxInt
1	2	3	4	5	6	7	8	9

Deterministic tree



maxInt	1	1	2	2	7	4	8	maxInt
1	2	3	4	5	6	7	8	9

Deterministic tree



maxInt	1	1	2	2	7	4	8	maxInt
1	2	3	4	5	6	7	8	9

Rest of the talk

- Second application of LVish to graphs
- Motivation for improvement
- BulkRetry

Maximal independent set

- Spawn a task for each vertex
- Check if any *higher-priority* neighbor has been added

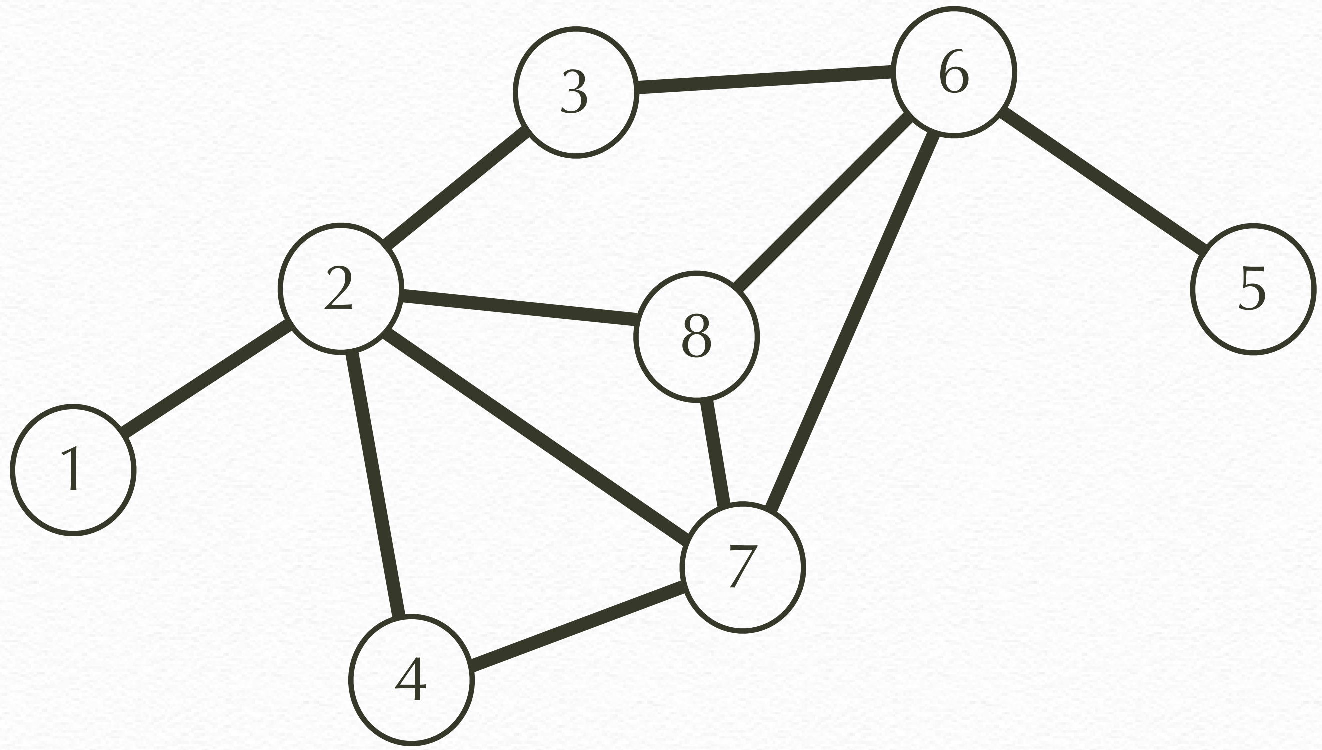
Maximal independent set

- Priorities handled using the blocking `get`

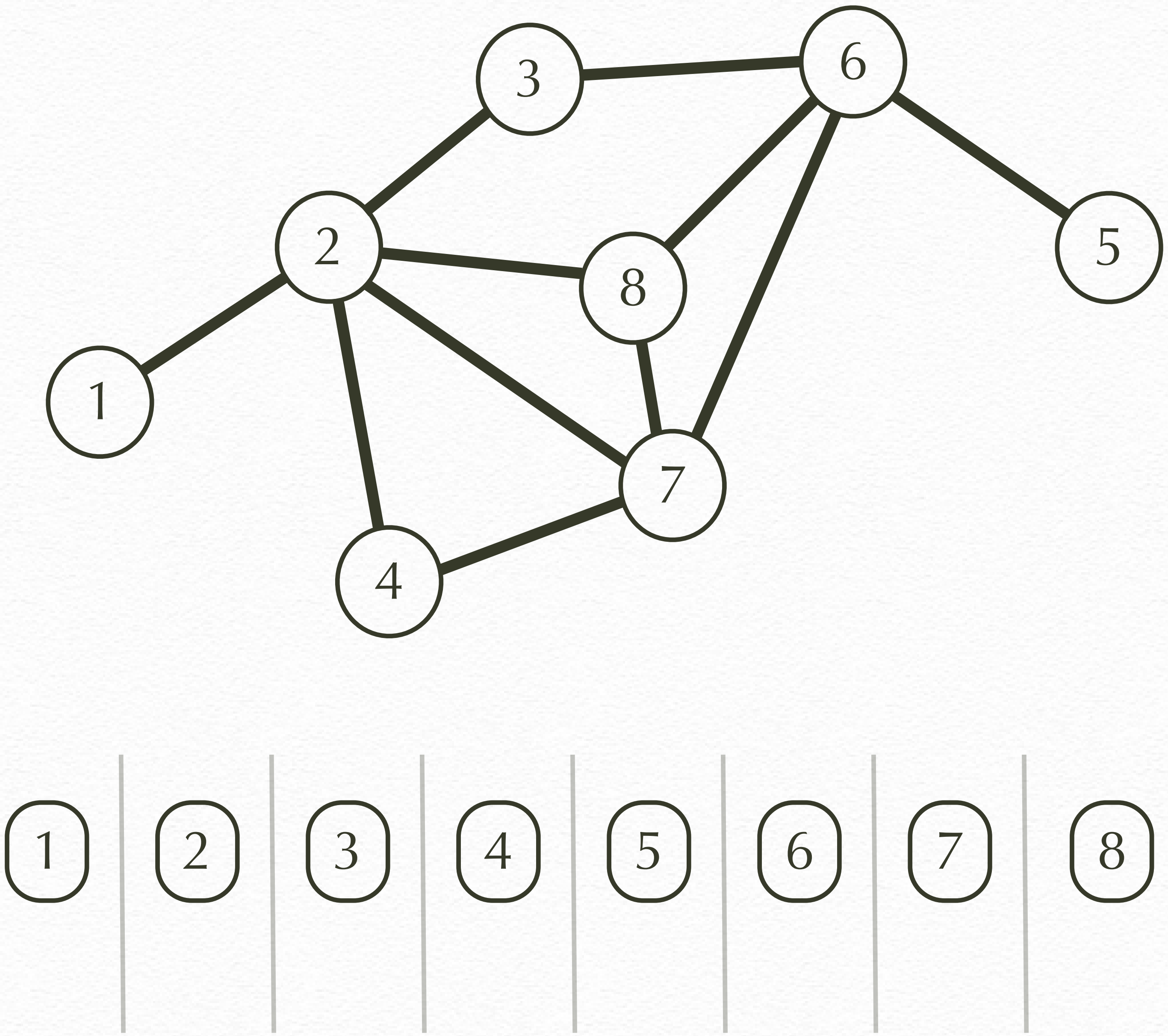
```
get flagsArr nbr

data Flag = Chosen | NbrChosen
```

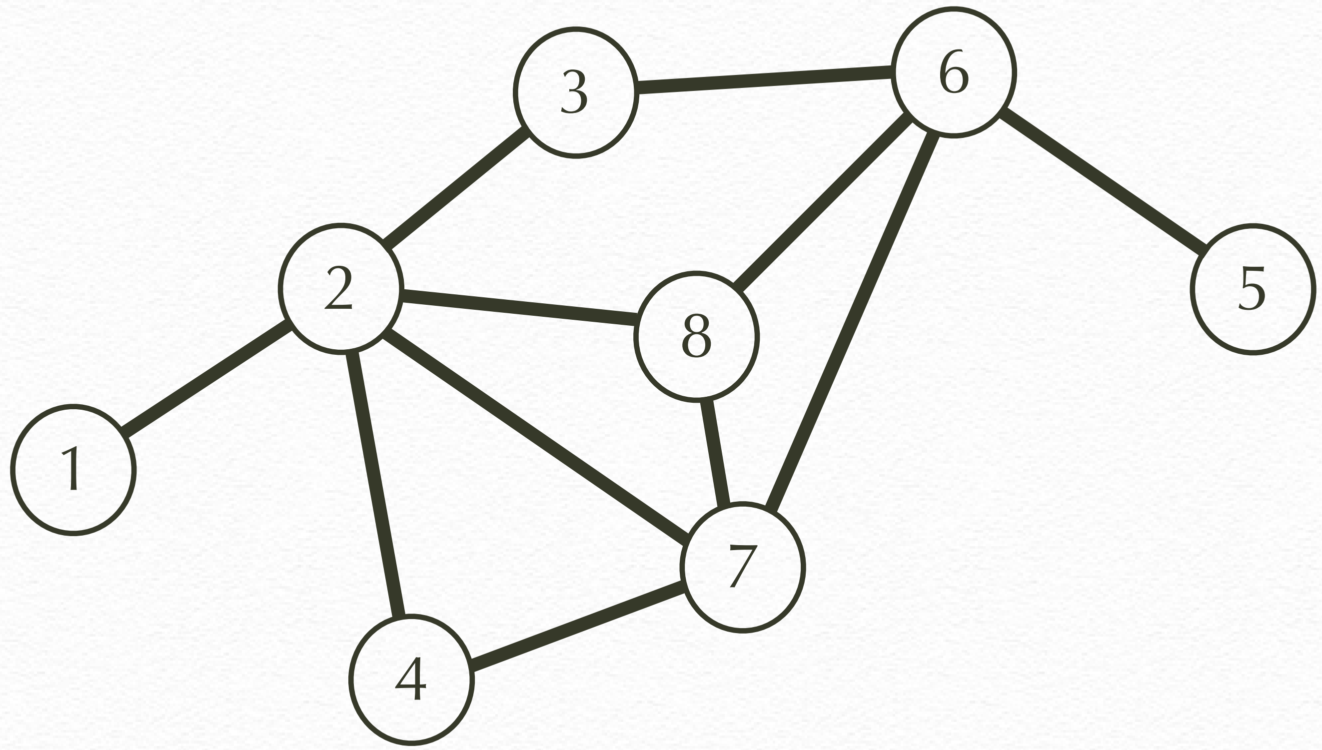
Maximal independent set



Maximal independent set

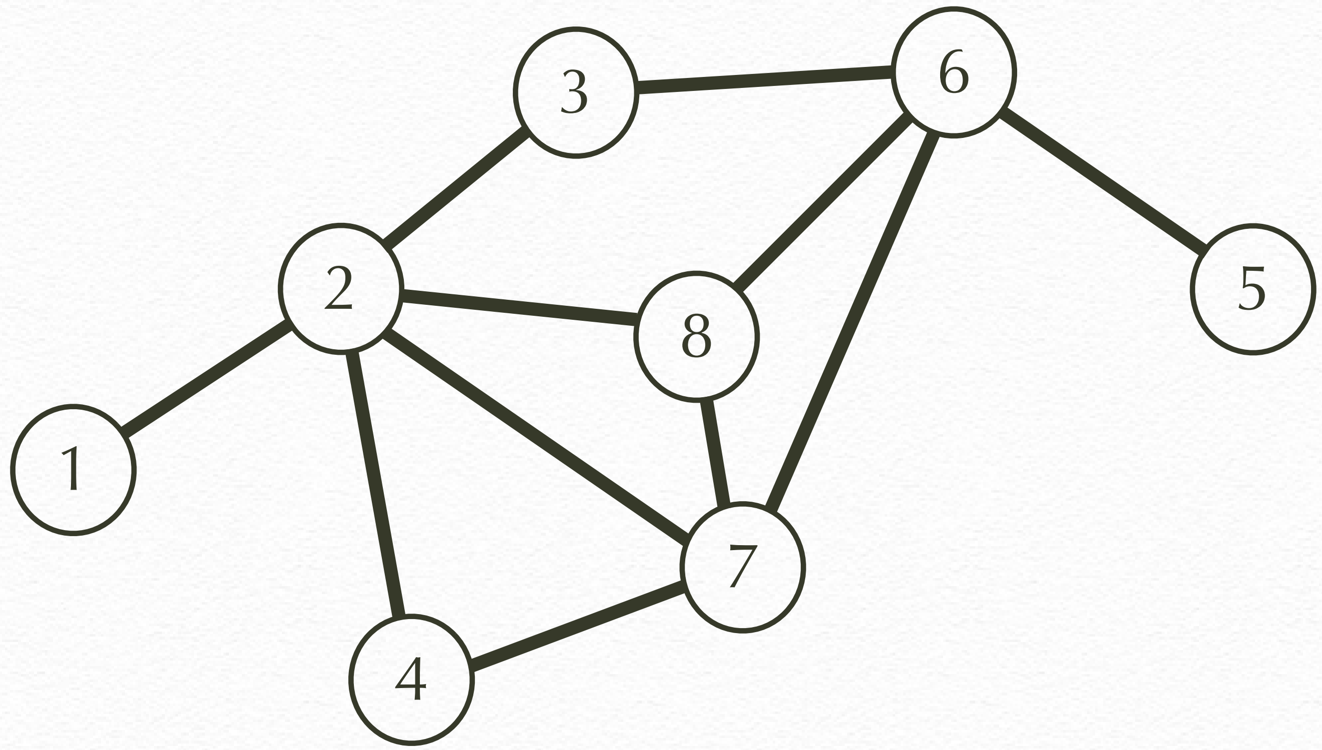


Maximal independent set



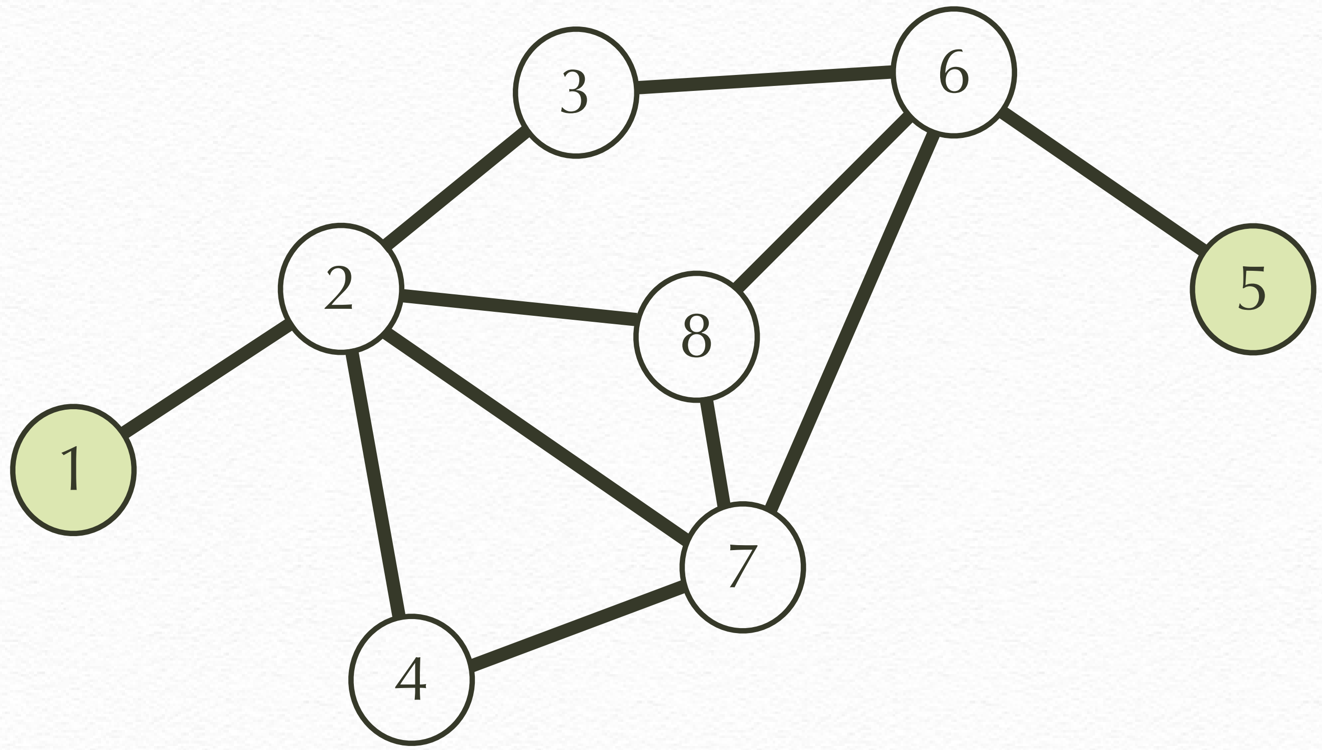
1	2	3	4	5	6	7	8
{}	{1}	{2}	{2}	{}	{3,5}	{2,4}	{2,6,7}

Maximal independent set



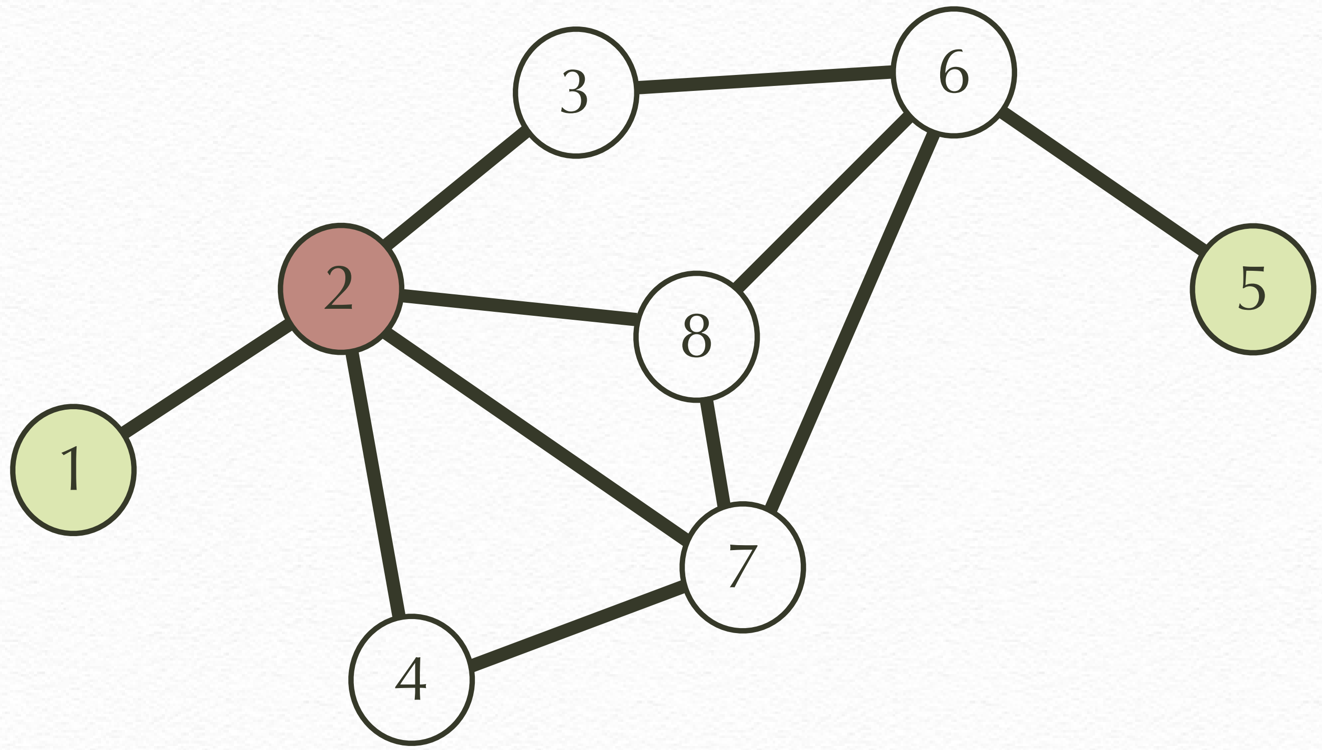
1	2	3	4	5	6	7	8
{}	{1}	{2}	{2}	{}	{3,5}	{2,4}	{2,6,7}

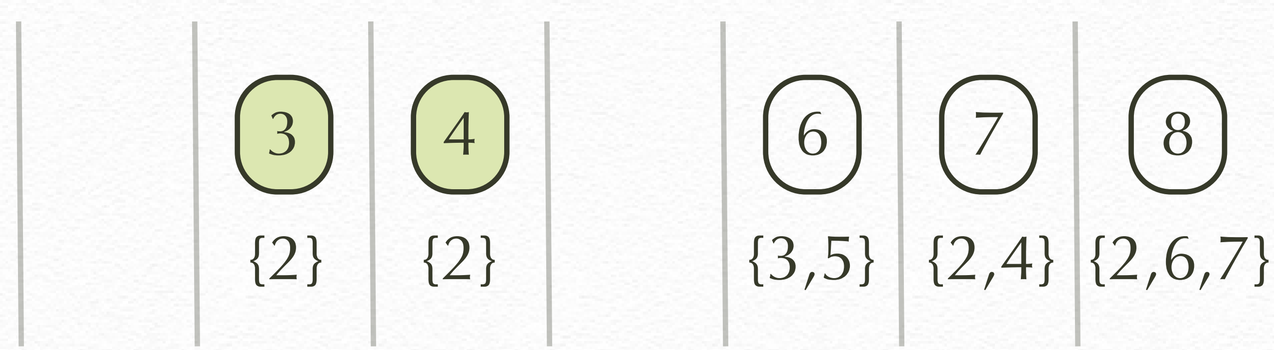
Maximal independent set



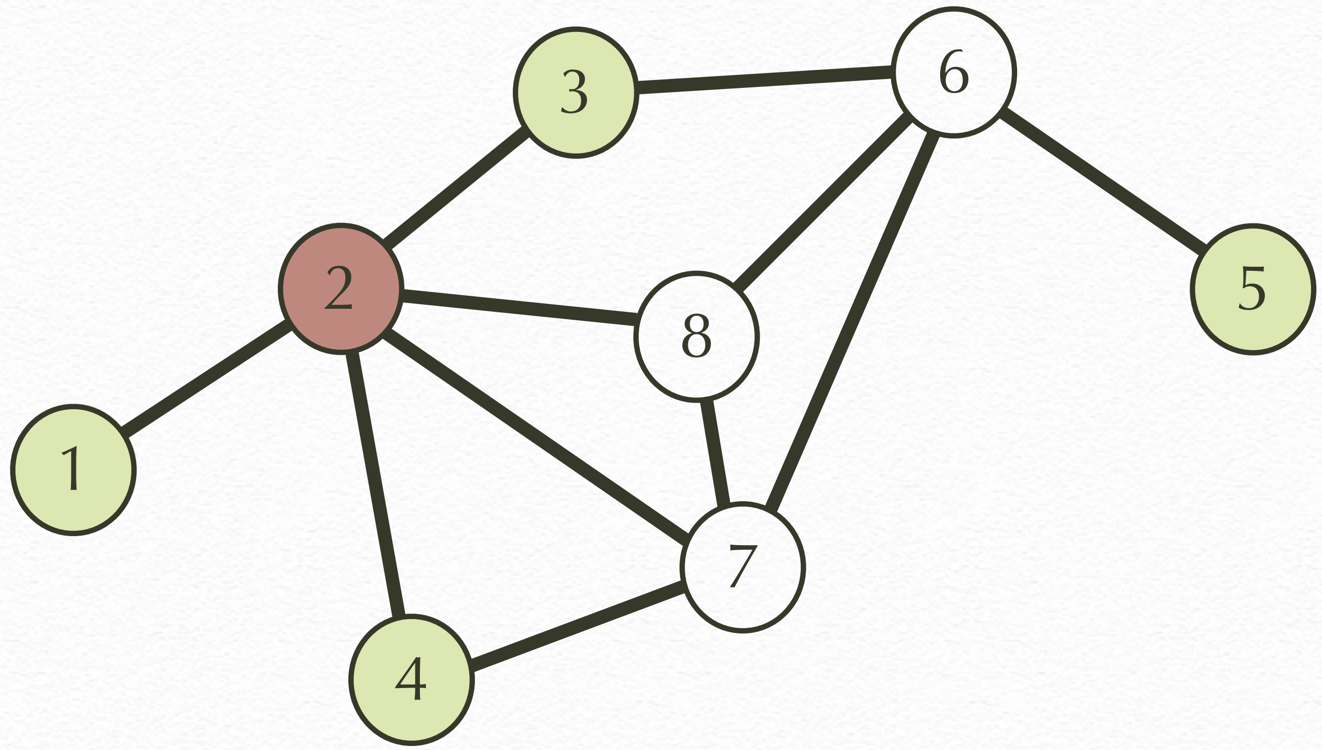
2	3	4		6	7	8
{1}	{2}	{2}		{3,5}	{2,4}	{2,6,7}

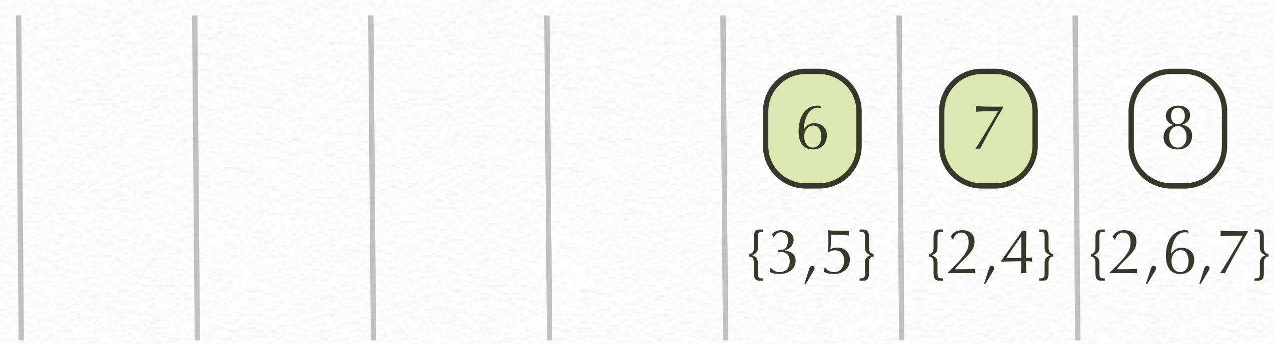
Maximal independent set



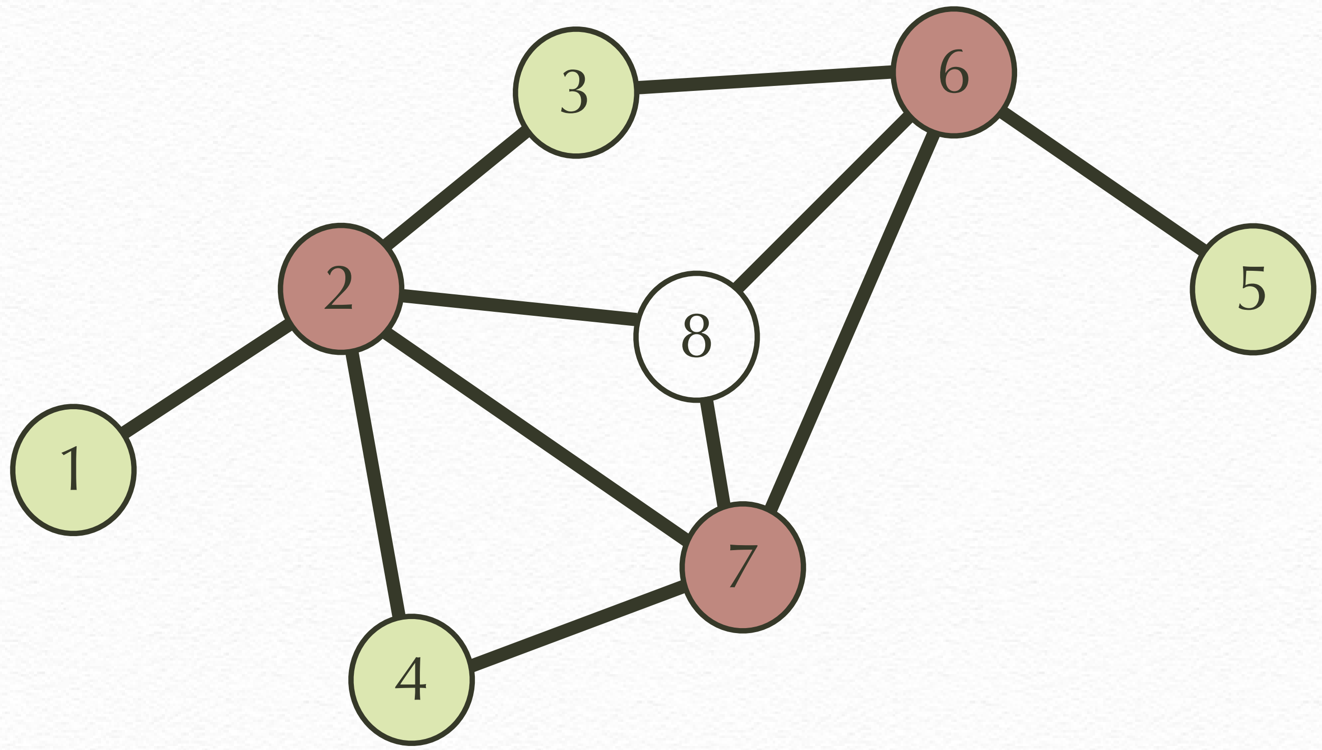


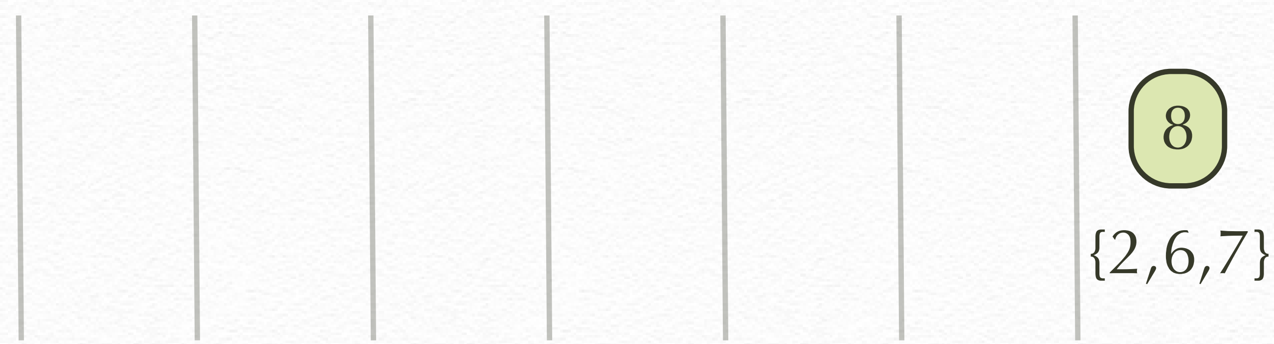
Maximal independent set



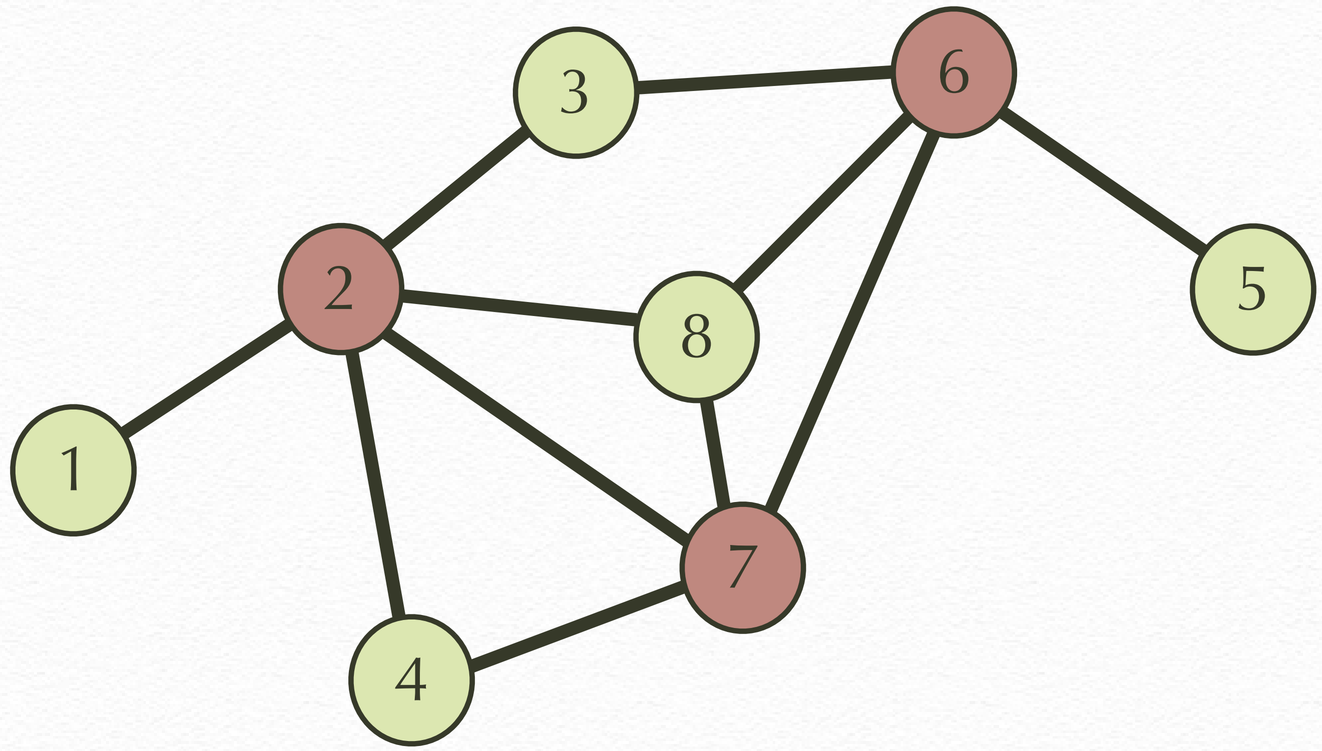


Maximal independent set





Maximal independent set



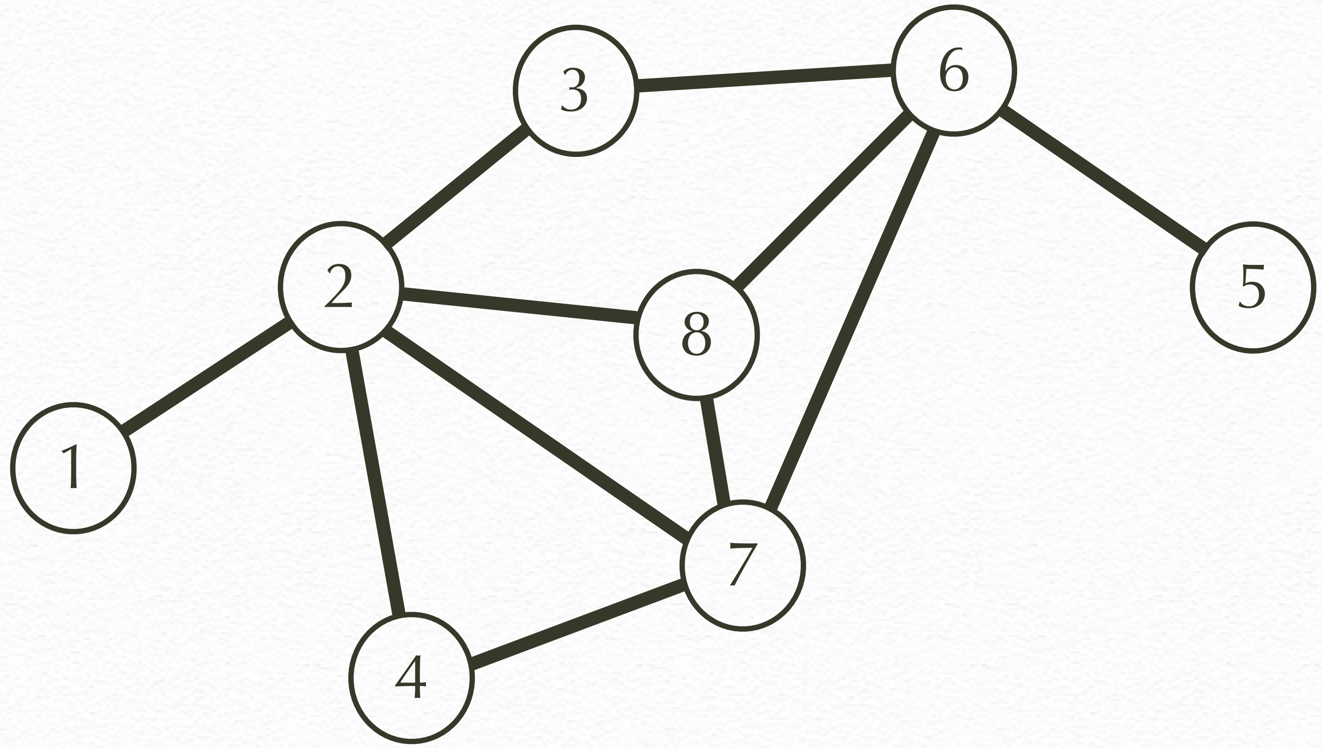
Observe

- Inefficient for large $|V|$
- Per vertex interaction with task scheduler

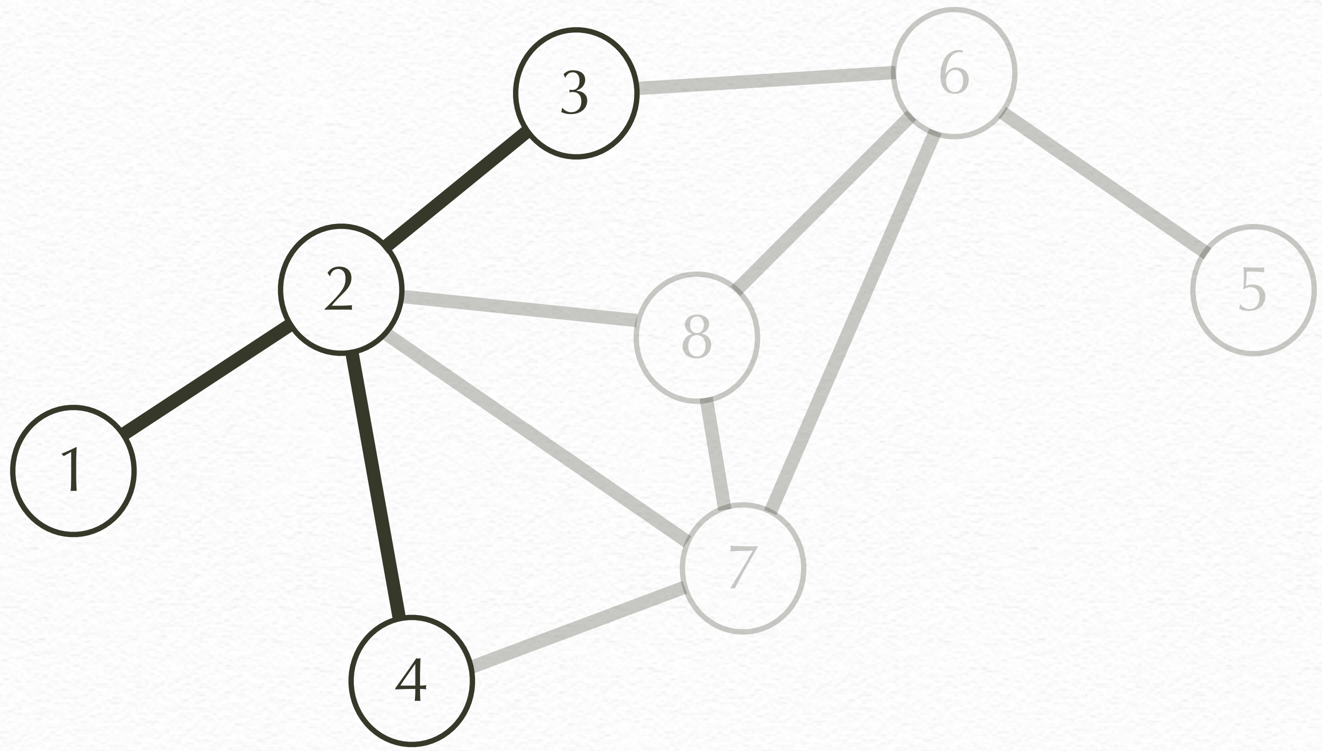
Deterministic Reservations

- Loop carried dependencies
- "Internally Deterministic Parallel Algorithms Can Be Fast", Blelloch *et al.*, *'12*

Deterministic Reservations

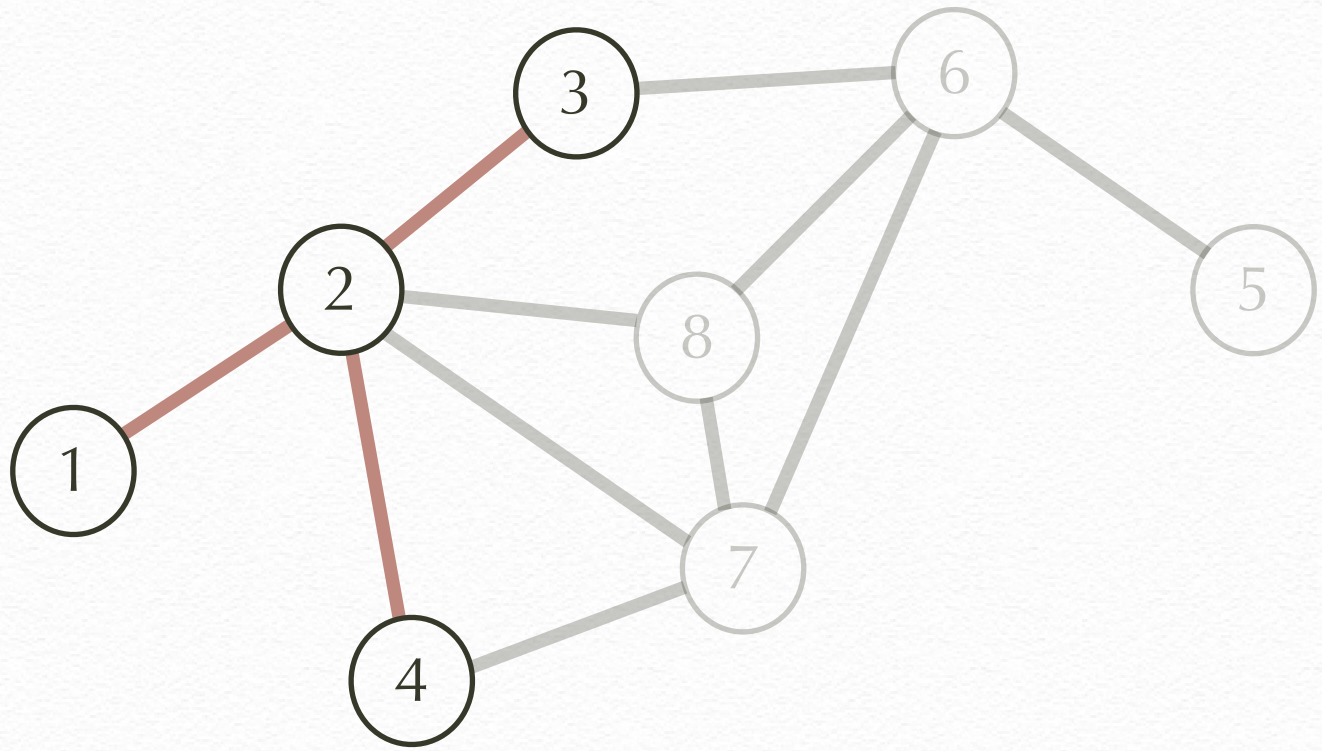


Deterministic Reservations



Round 1

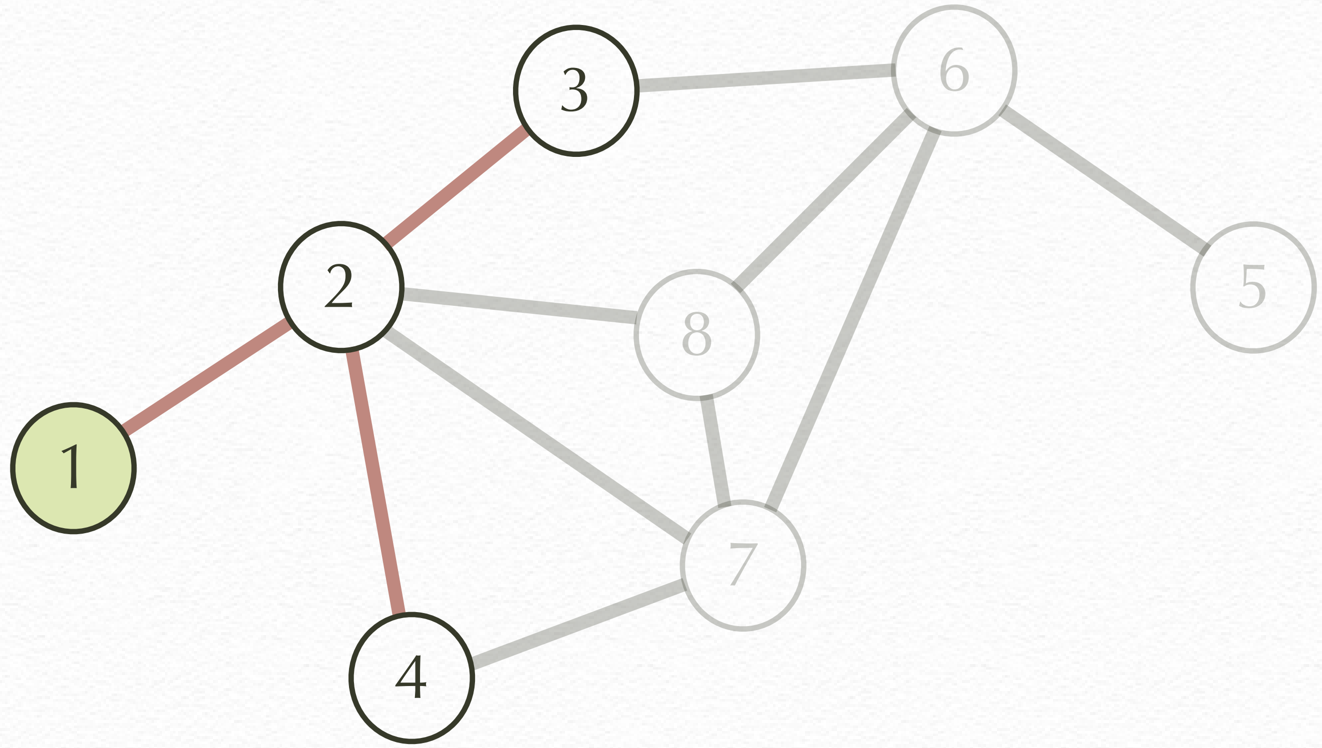
Deterministic Reservations



Round 1

Reserve Phase

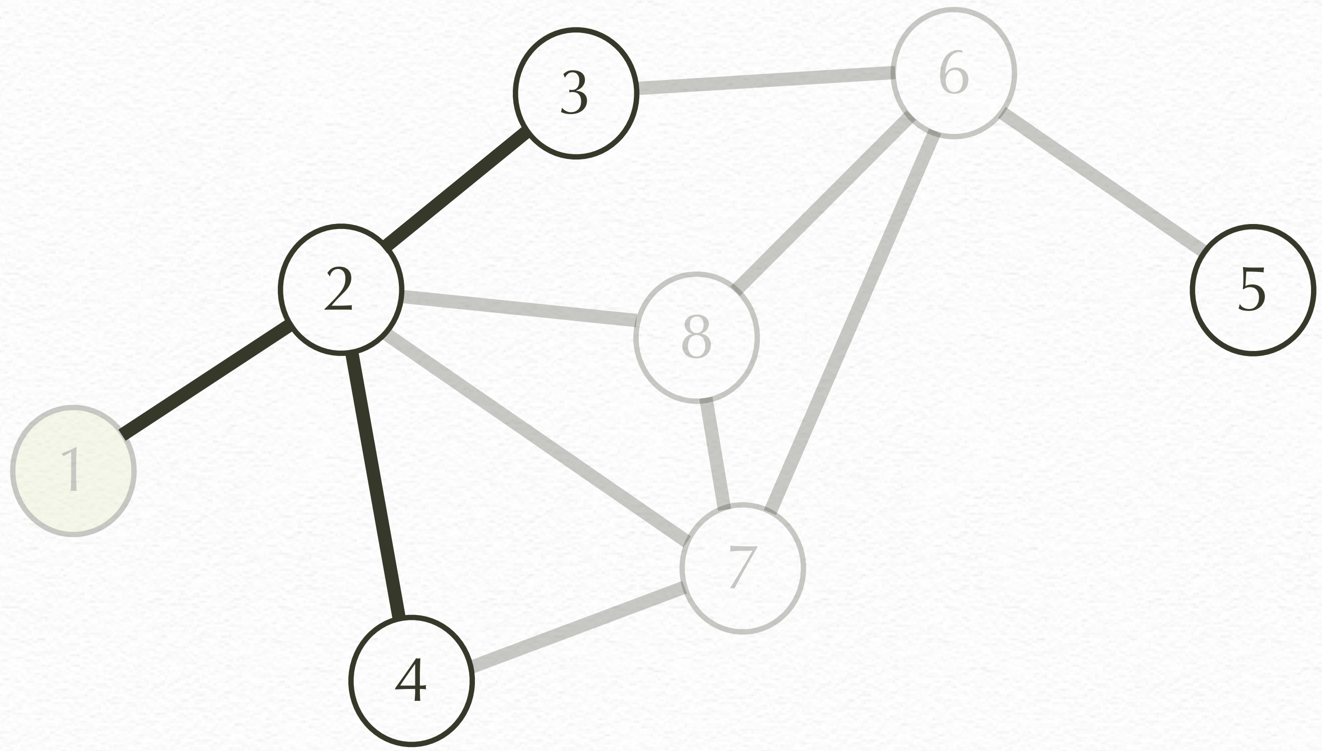
Deterministic Reservations



Round 1

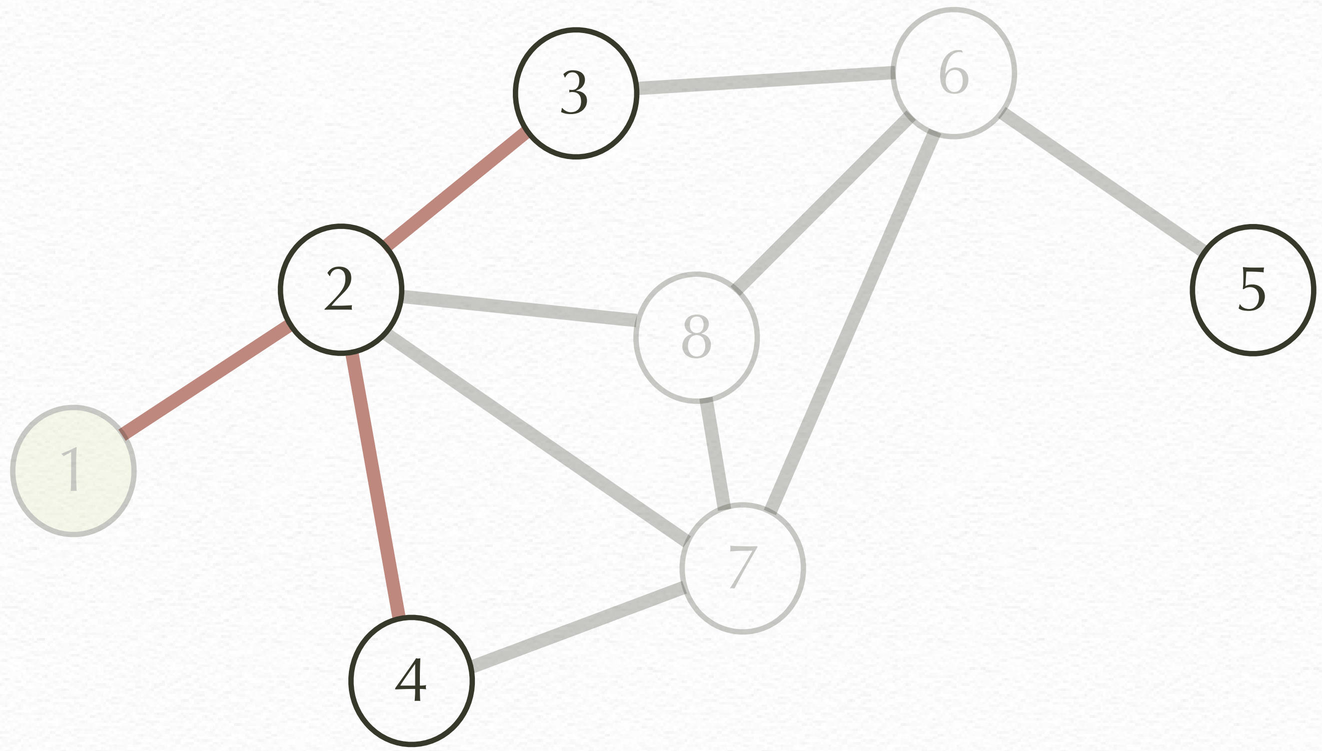
Commit Phase

Deterministic Reservations



Round 2

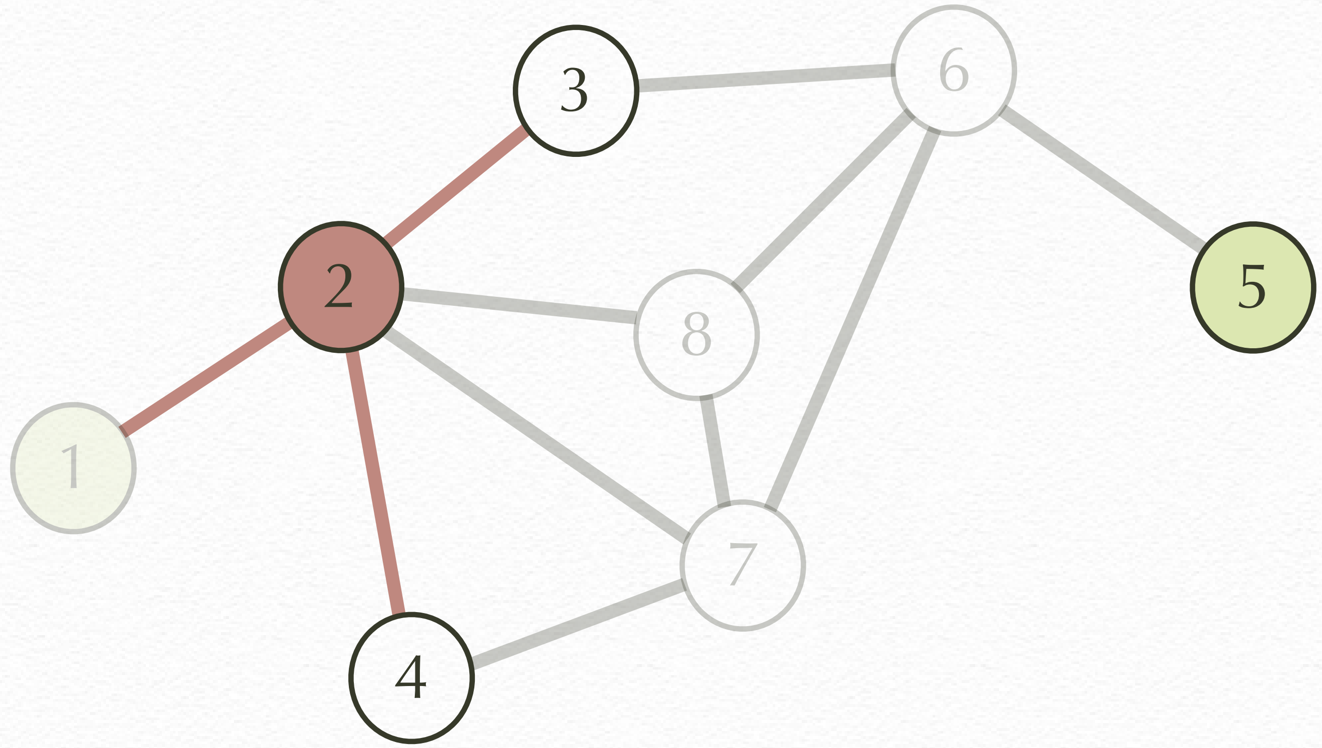
Deterministic Reservations



Round 2

Reserve Phase

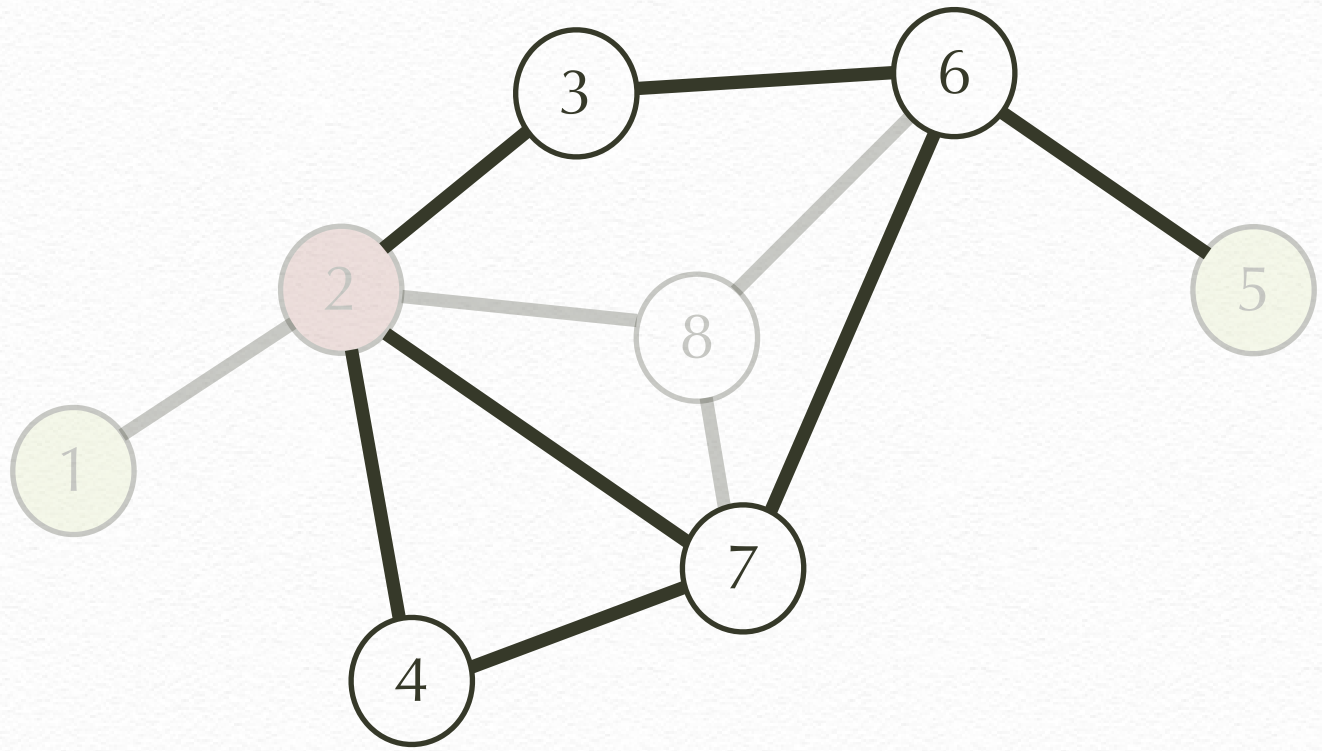
Deterministic Reservations



Round 2

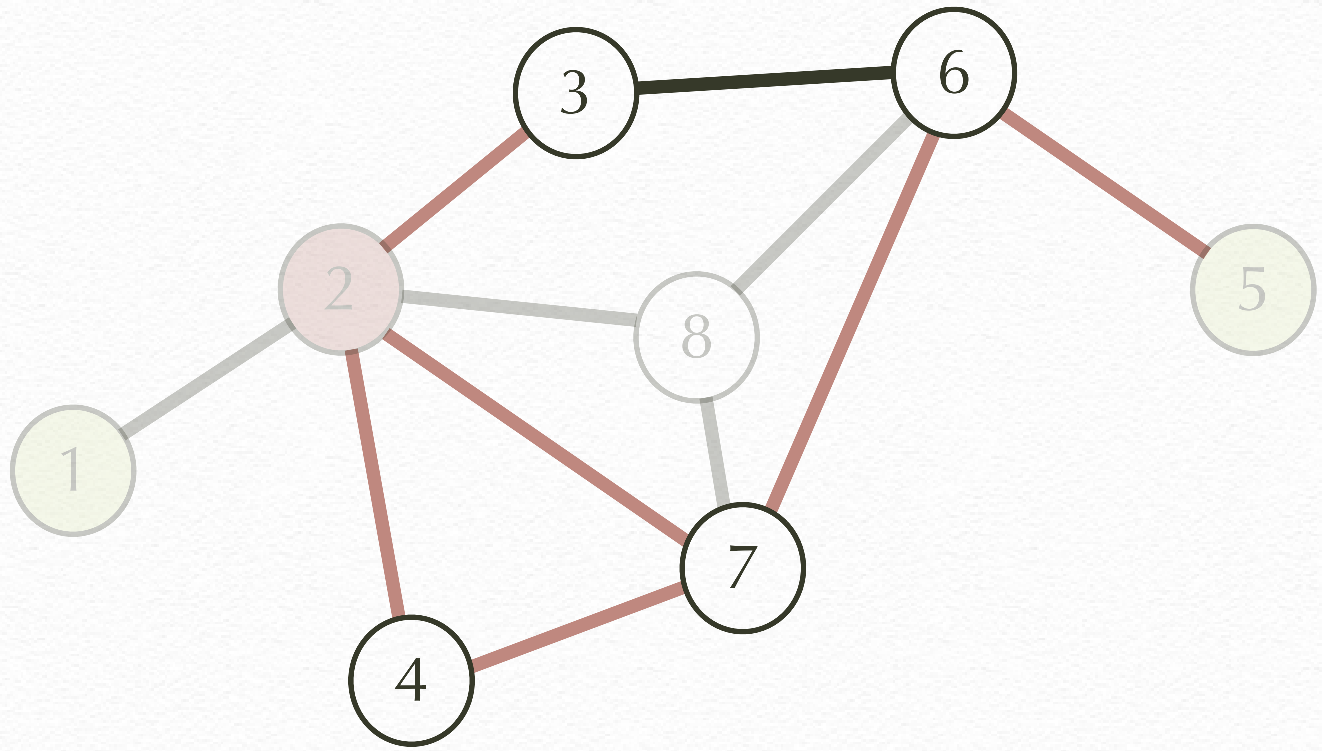
Commit Phase

Deterministic Reservations



Round 3

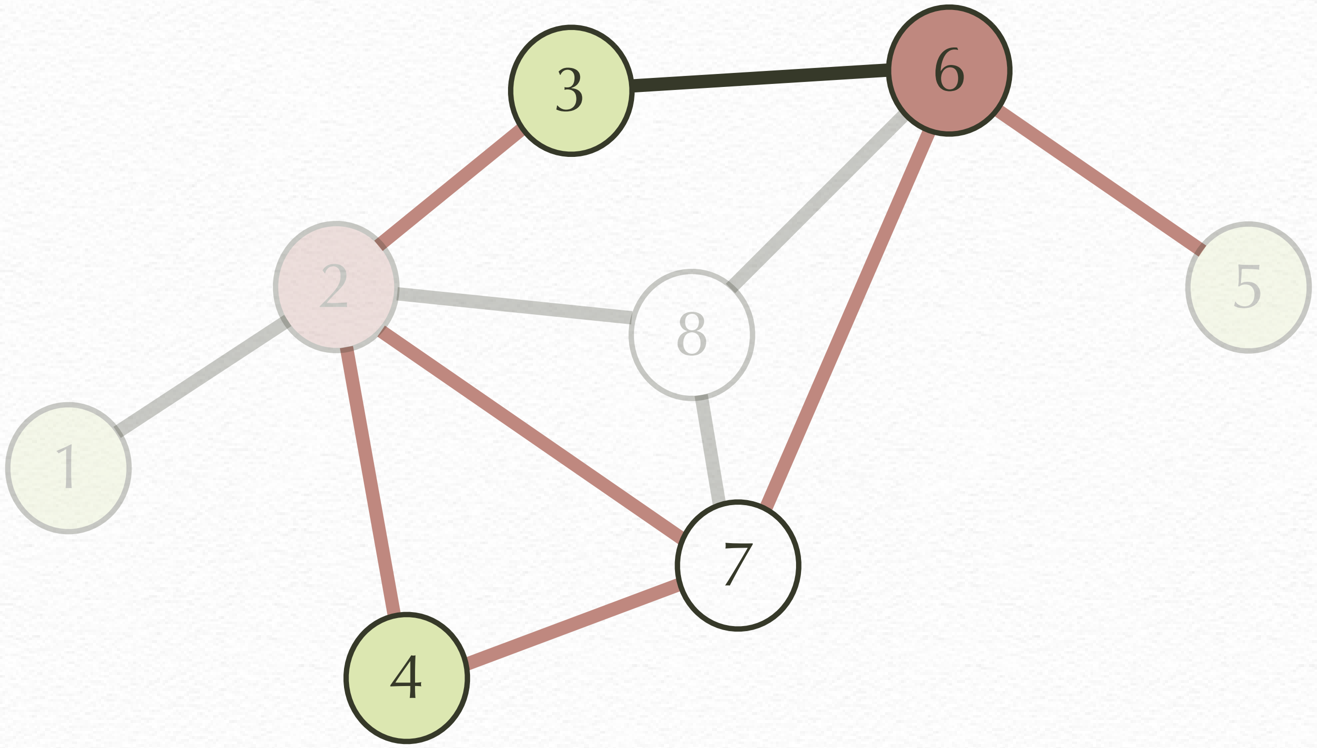
Deterministic Reservations



Round 3

Reserve Phase

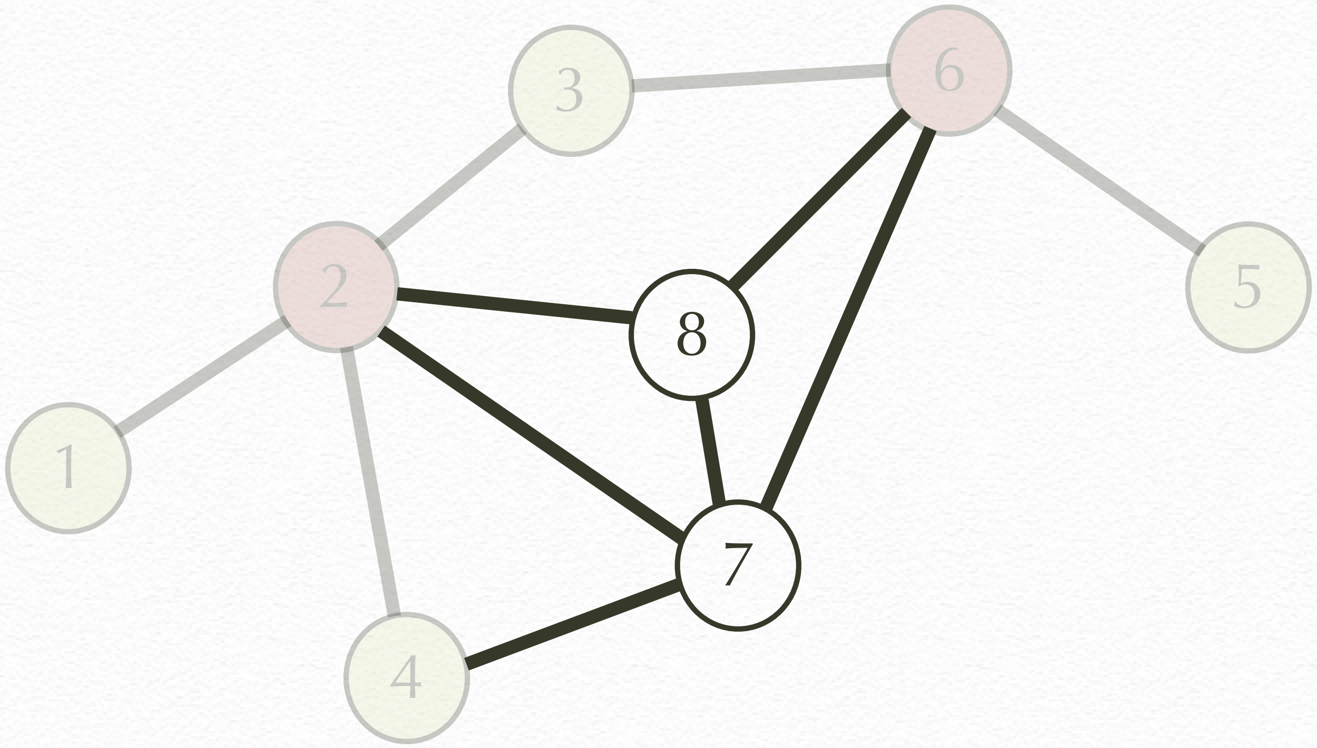
Deterministic Reservations



Round 3

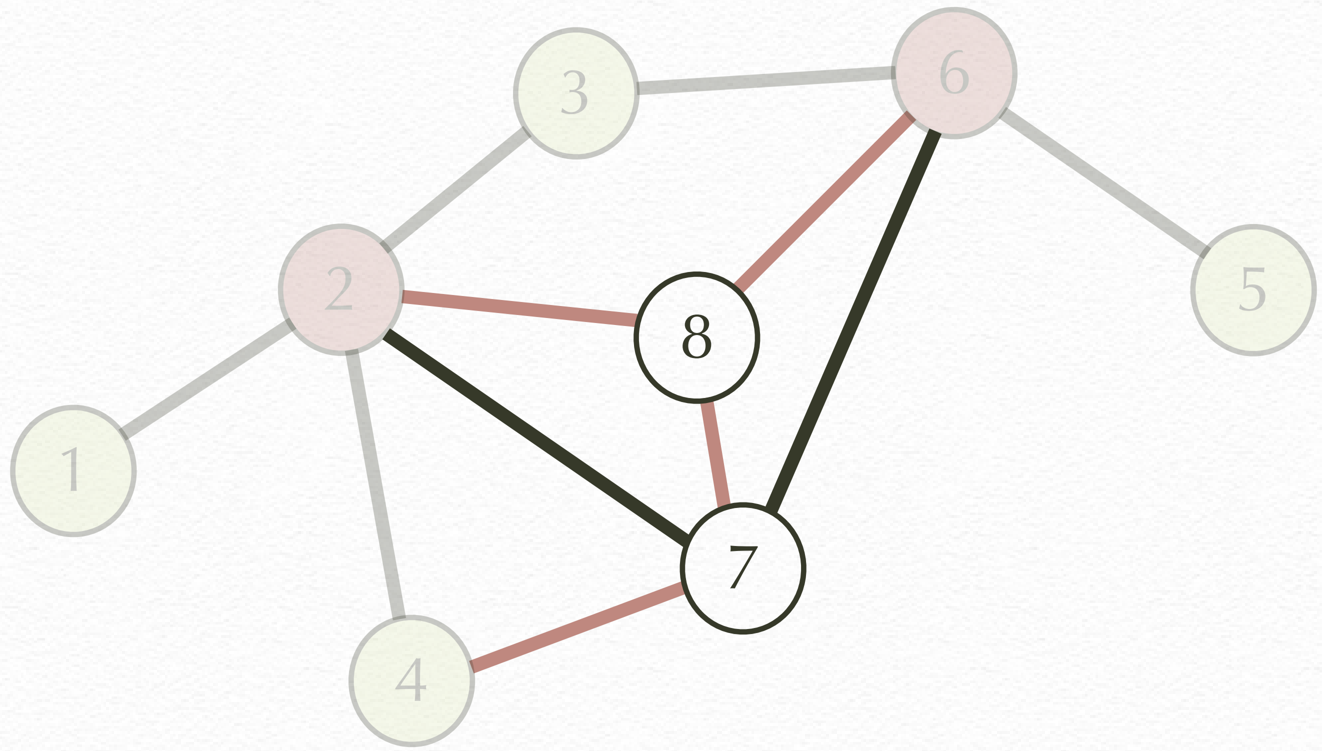
Commit Phase

Deterministic Reservations



Round 4

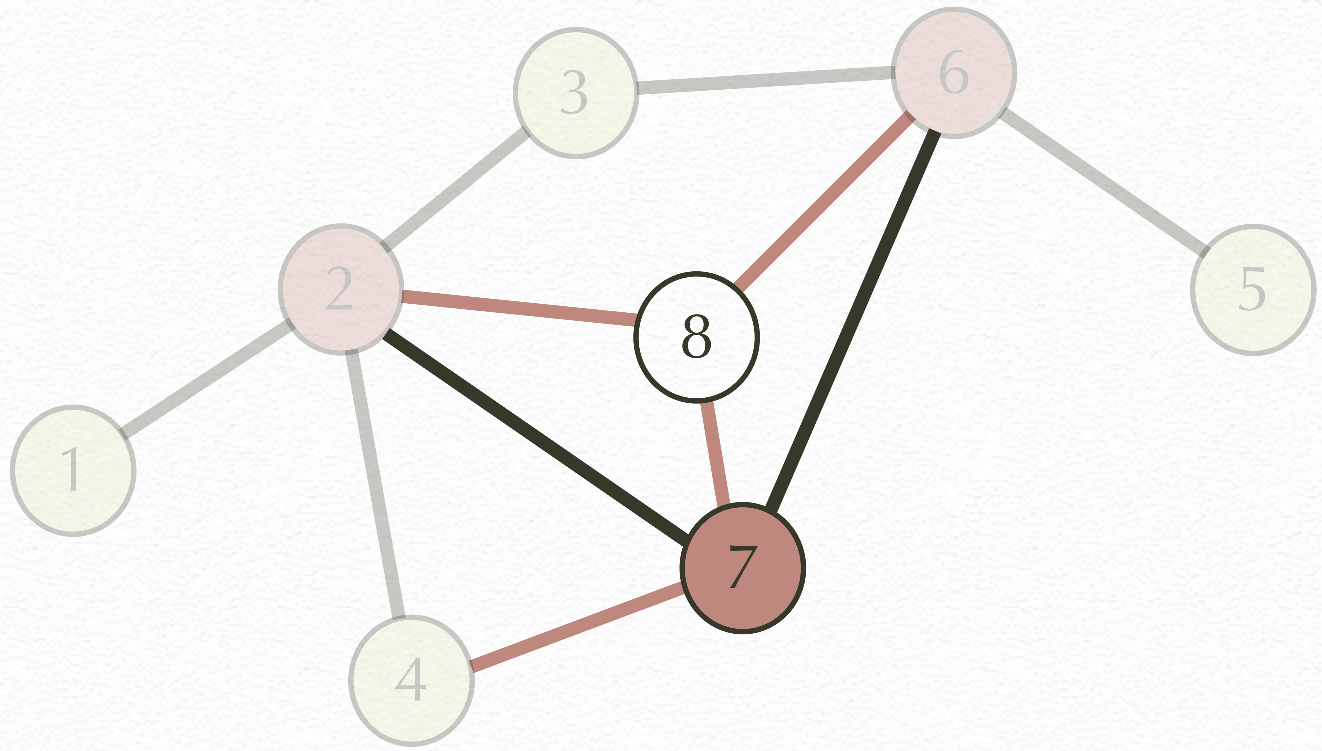
Deterministic Reservations



Round 4

Reserve Phase

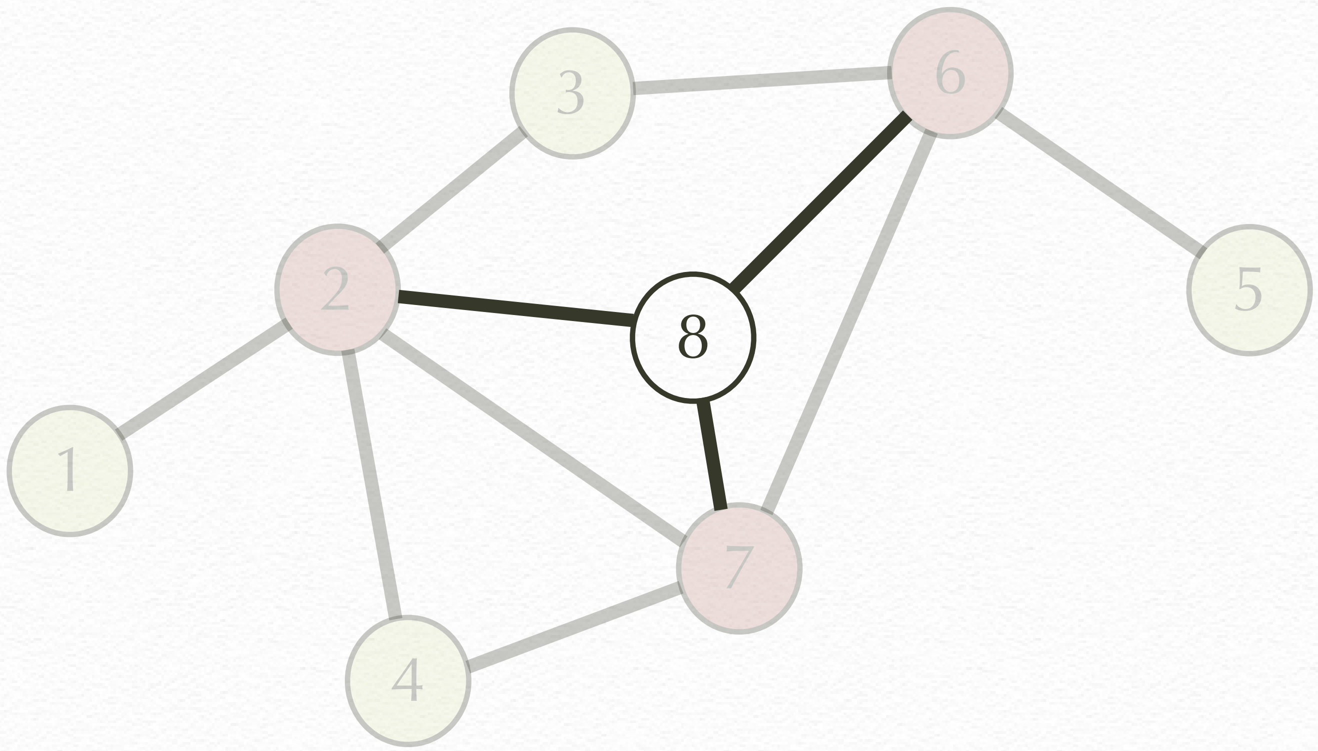
Deterministic Reservations



Round 4

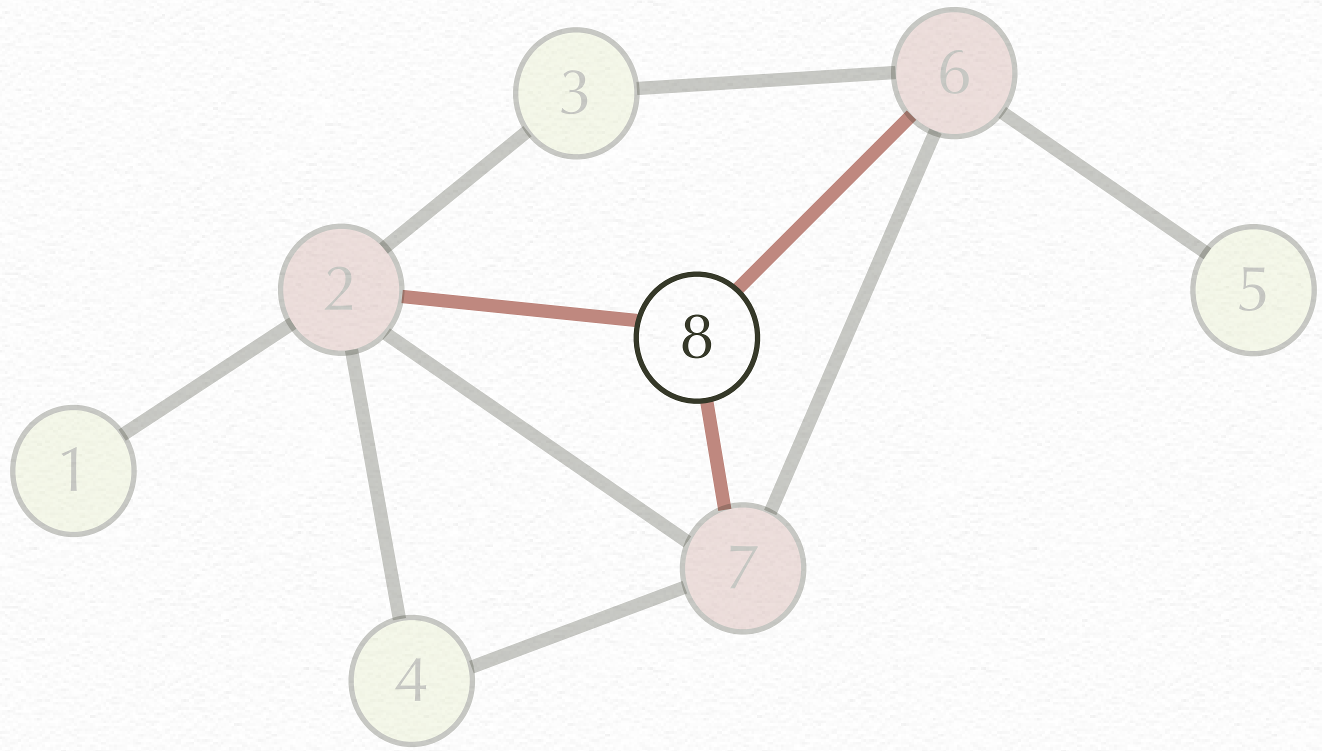
Commit Phase

Deterministic Reservations



Round 5

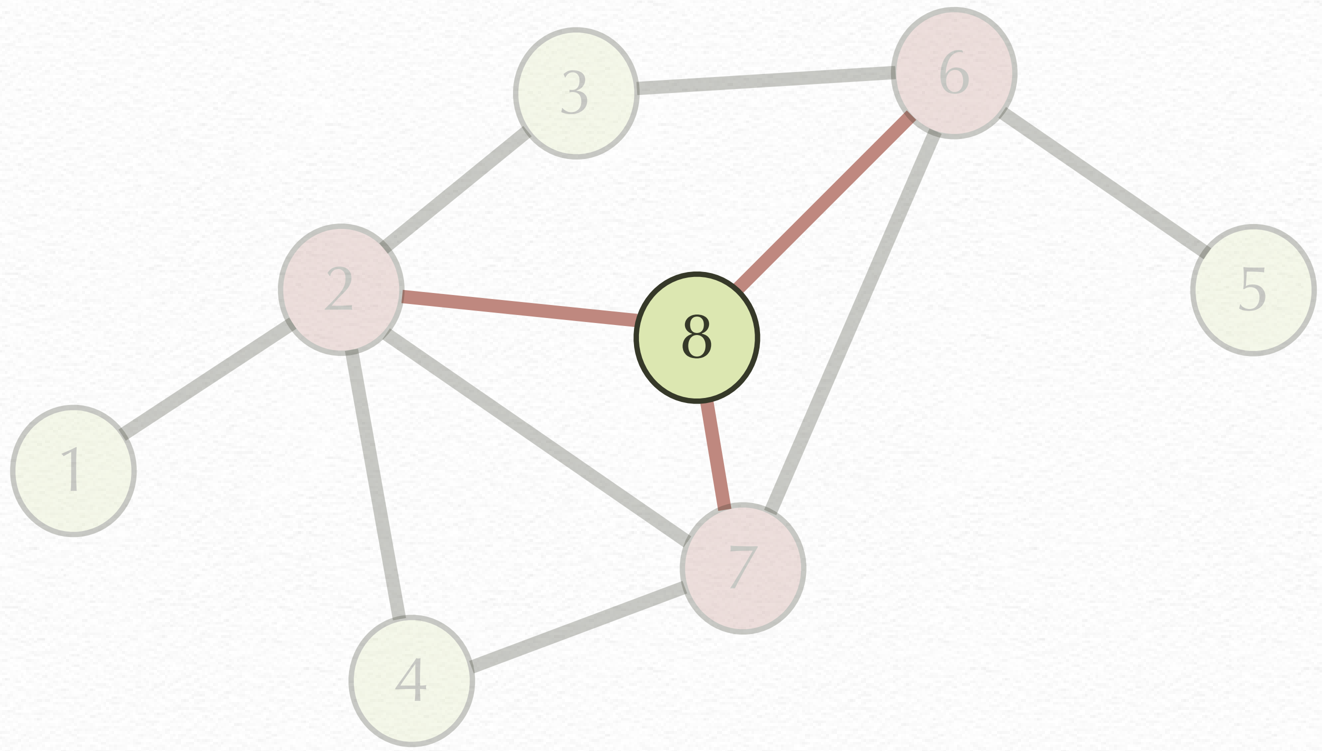
Deterministic Reservations



Round 5

Reserve Phase

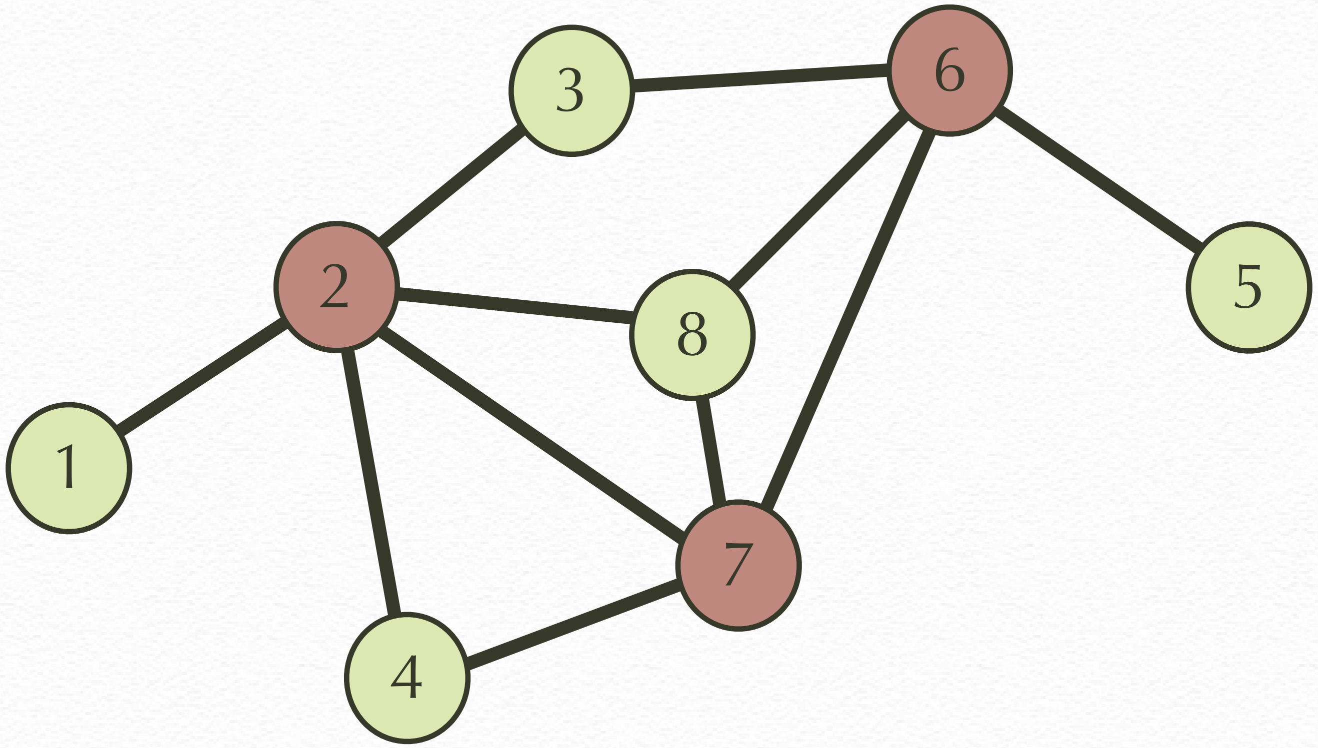
Deterministic Reservations



Round 5

Commit Phase

Deterministic Reservations



Lessons:

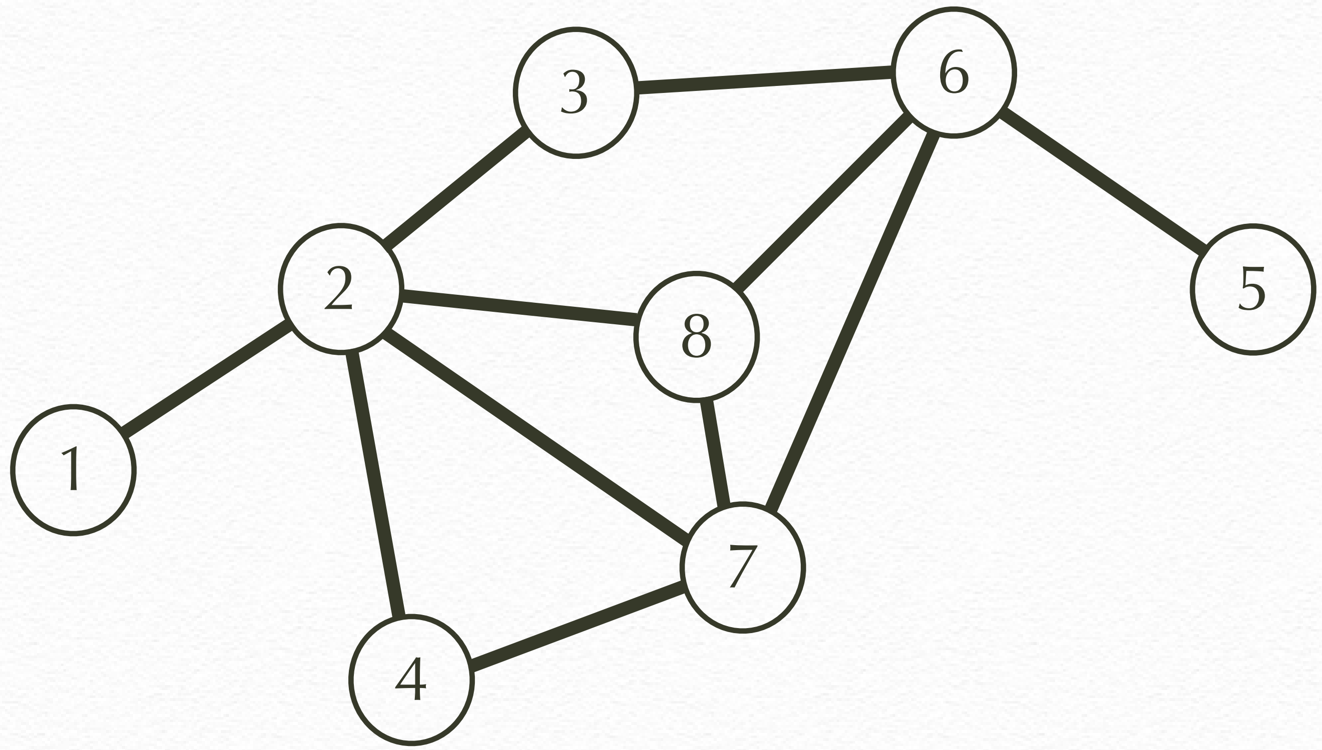
Bulk processing

Retry rather than blocking

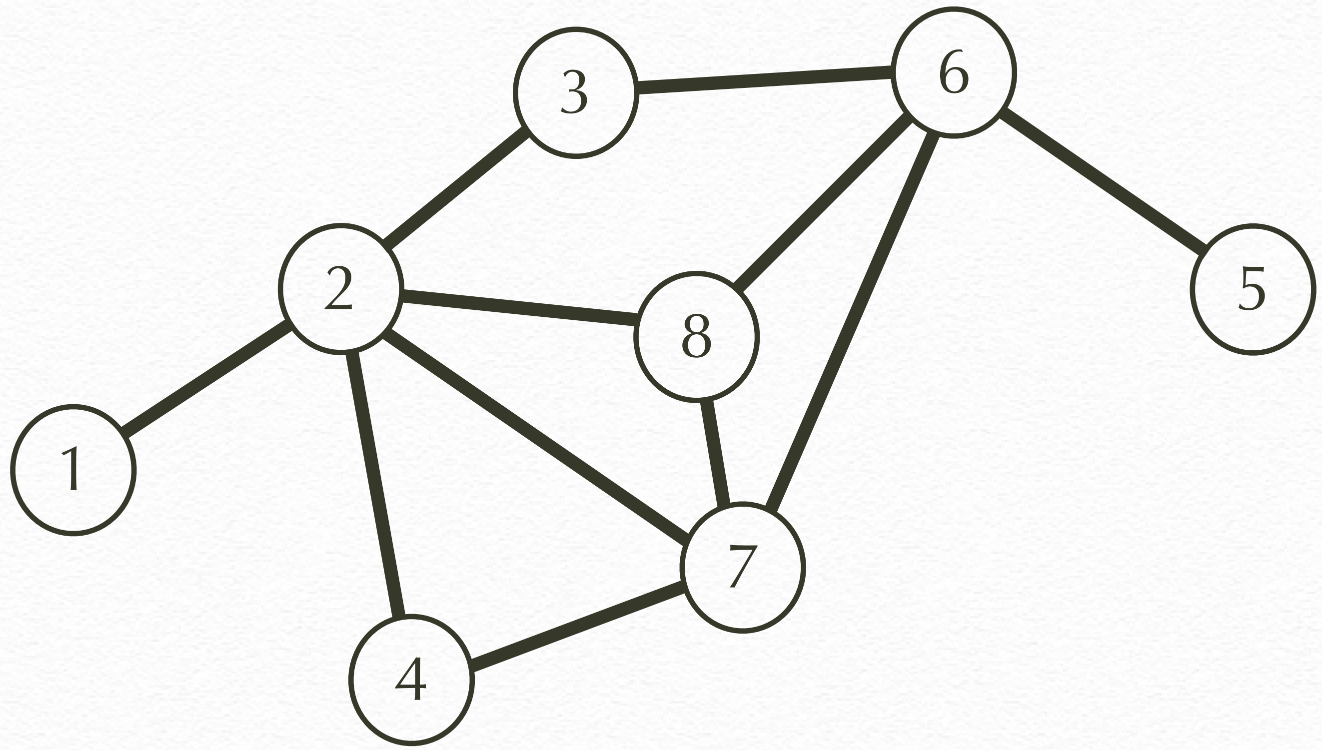
BulkRetry in LVish

- Bulk: For $n = pq$ iterations, spawn p tasks, each processing q iterations in sequence
- If the iteration dependency is not met, abort and retry later
- LVish operations are *idempotent*: retry and blocking have same semantics

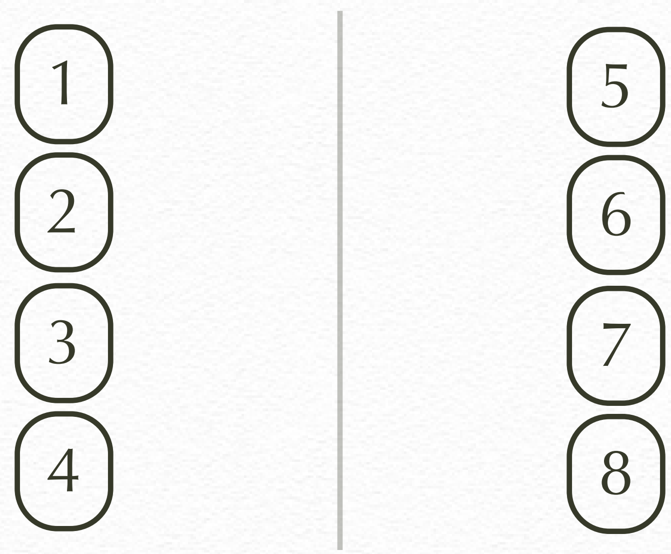
BulkRetry in LVish



BulkRetry in LVish

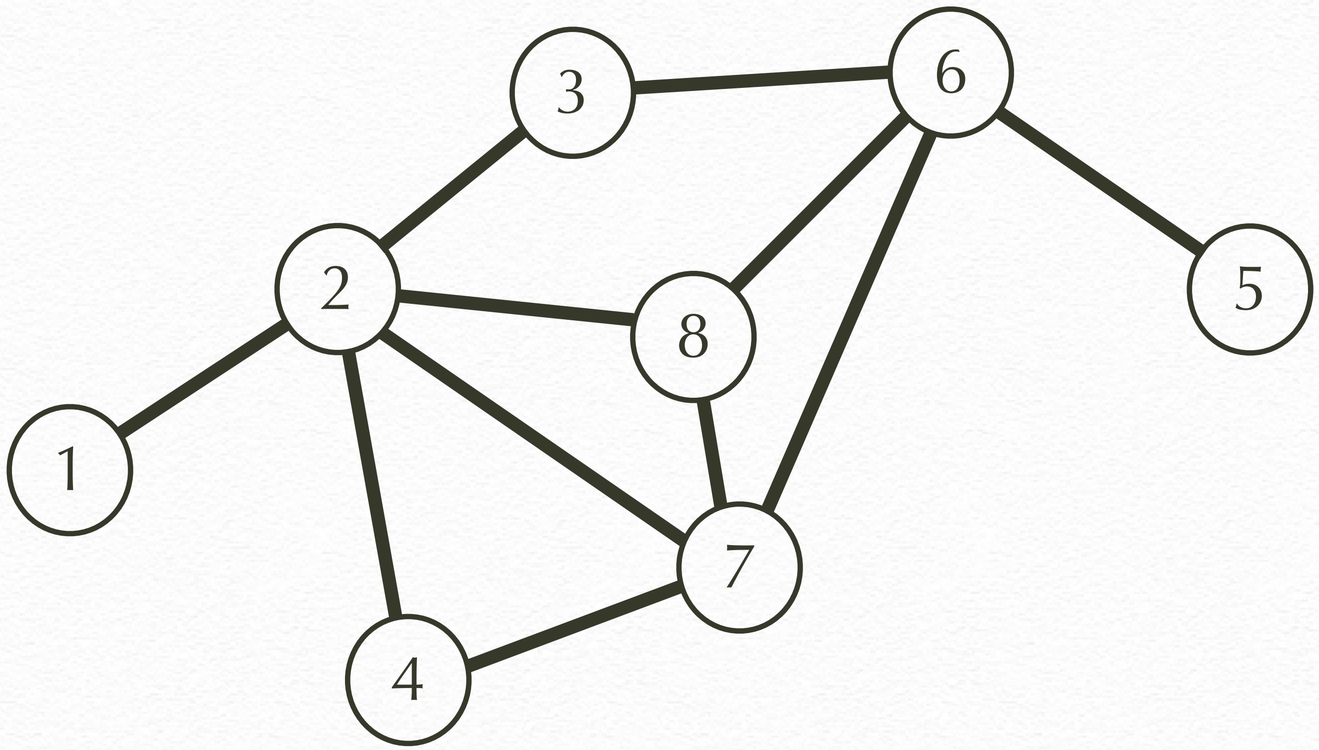


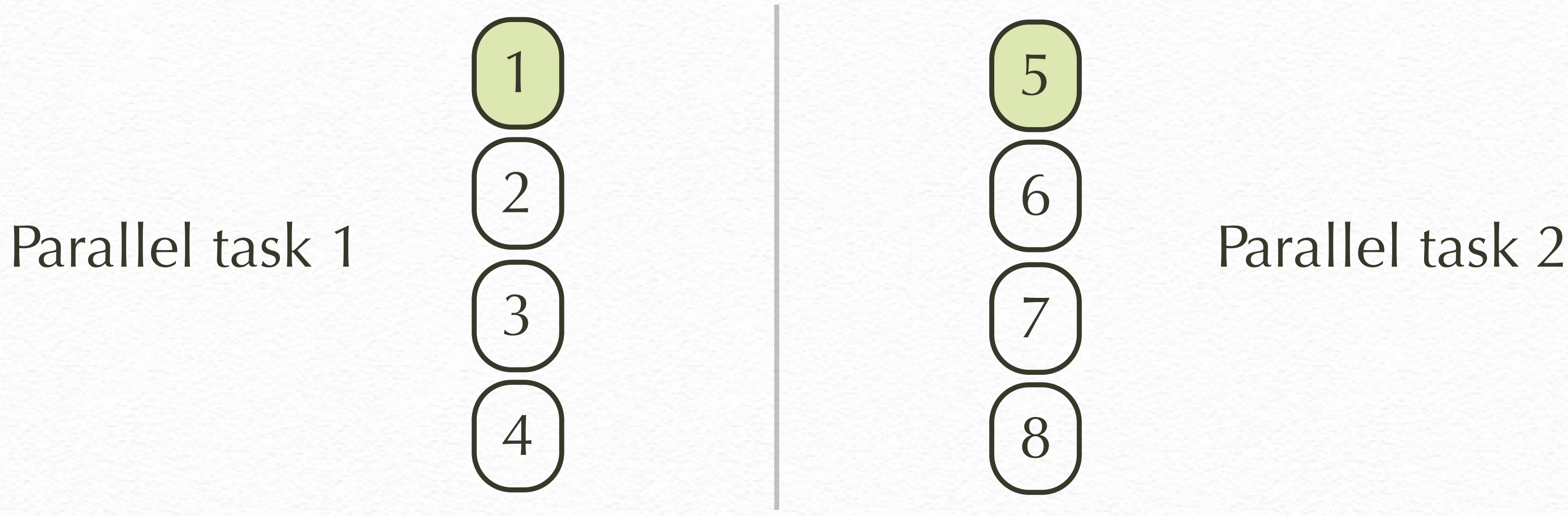
Parallel task 1



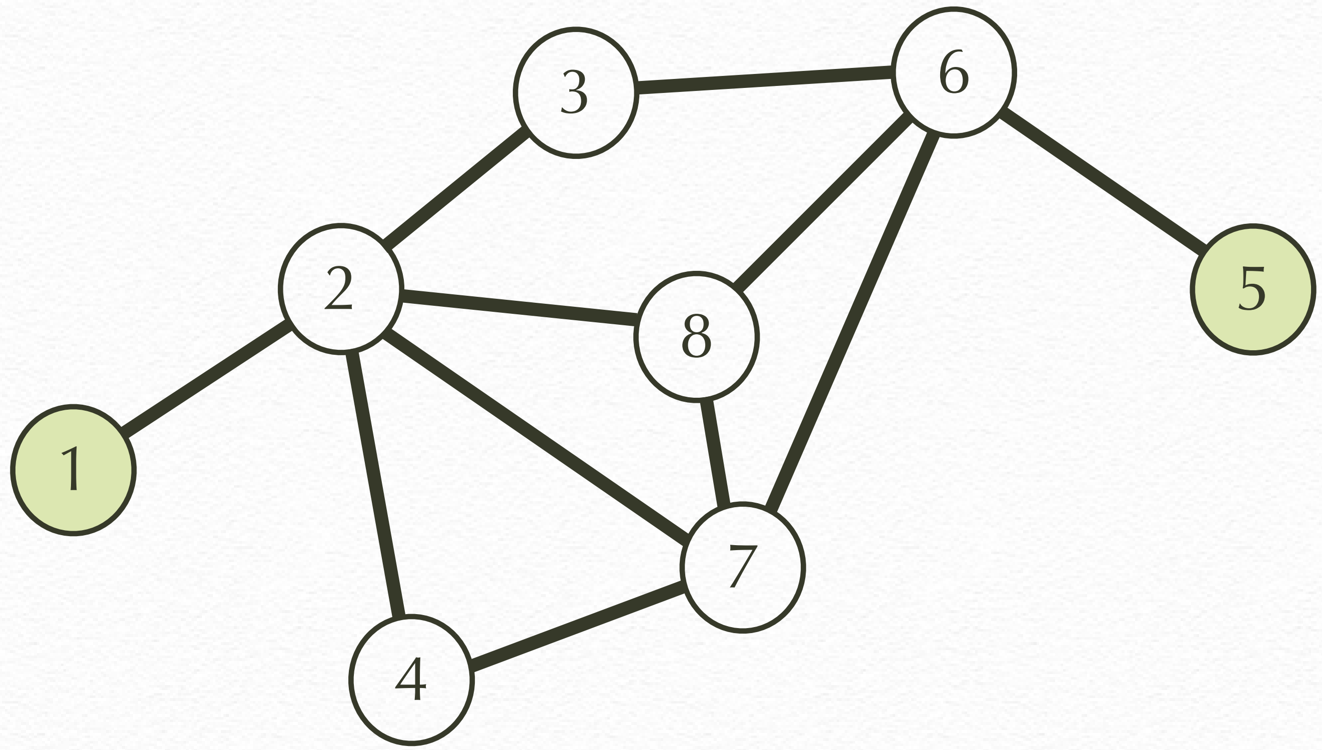
Parallel task 2

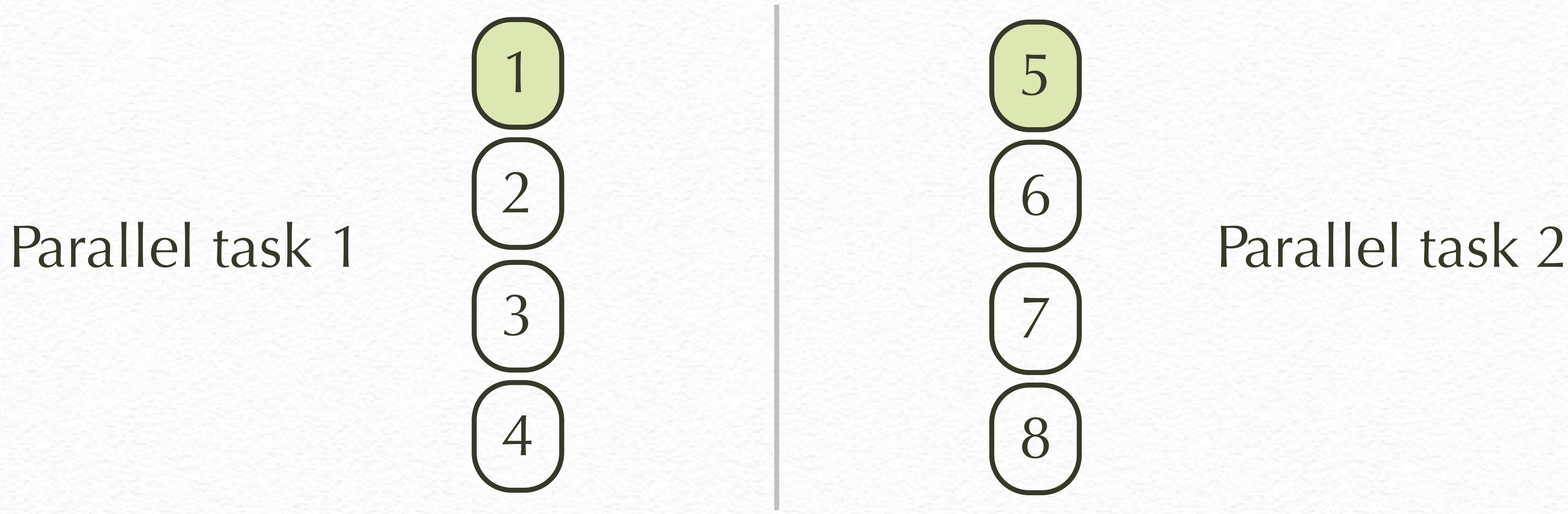
BulkRetry in LVish



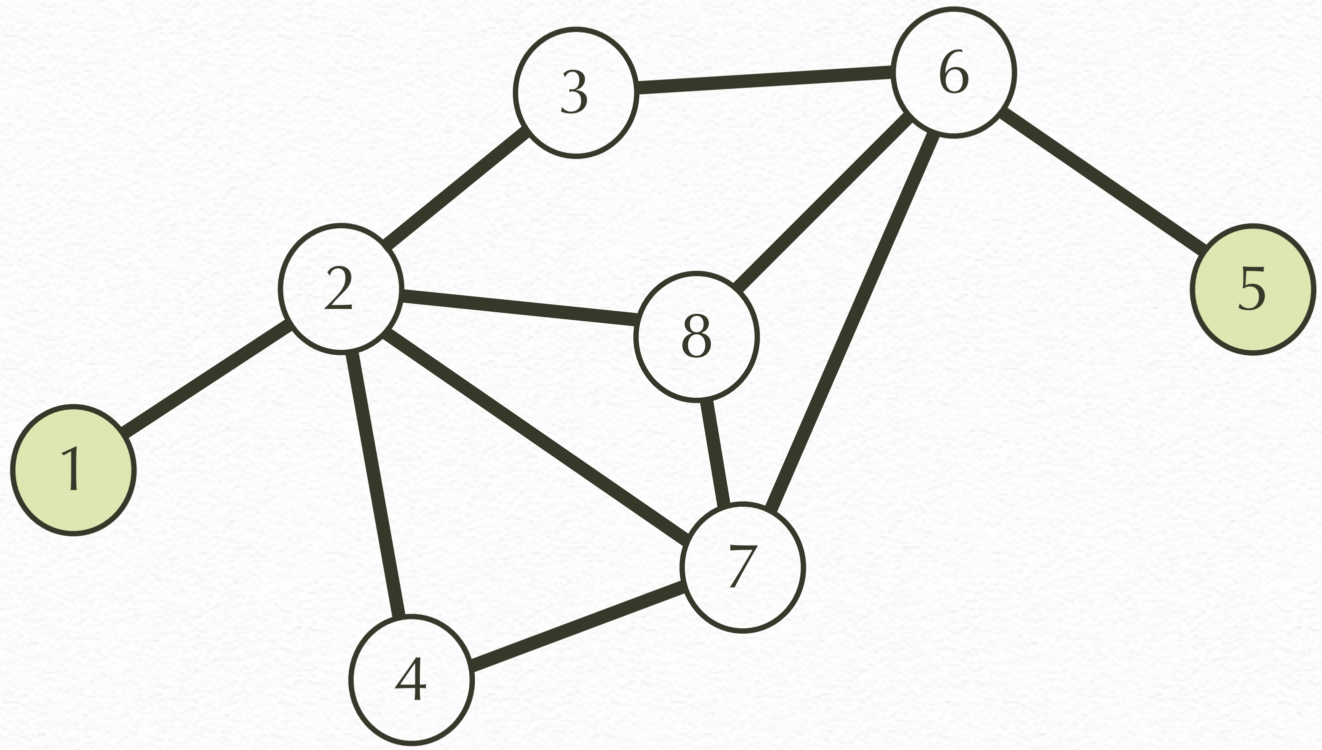


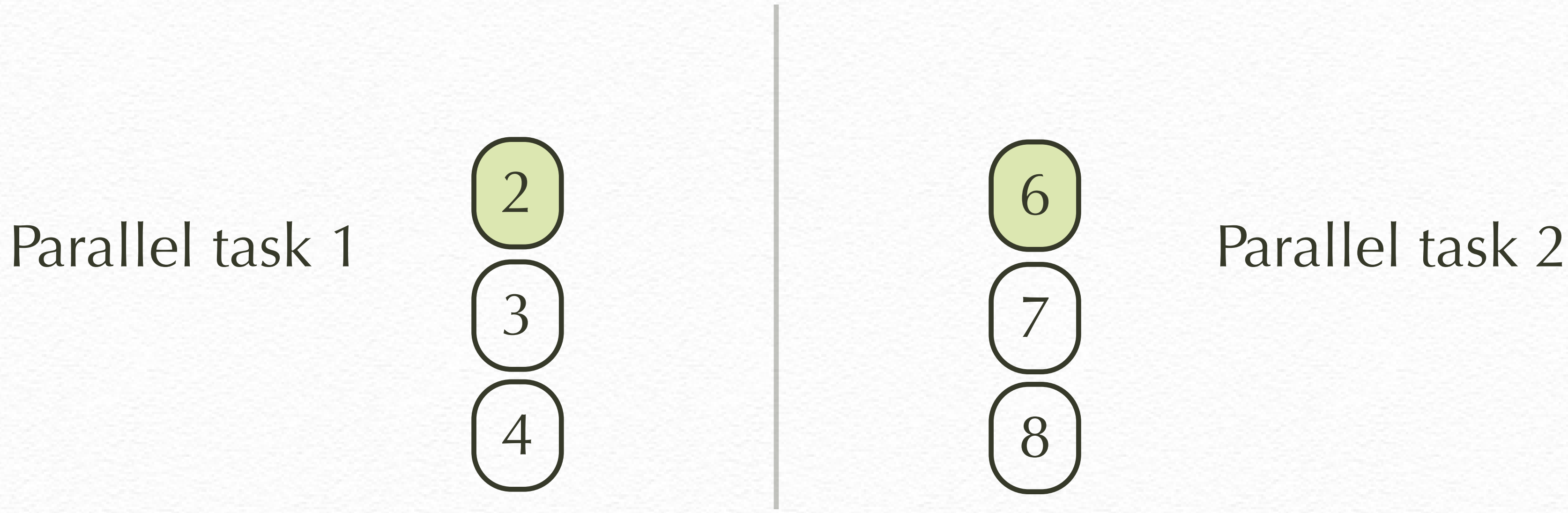
BulkRetry in LVish



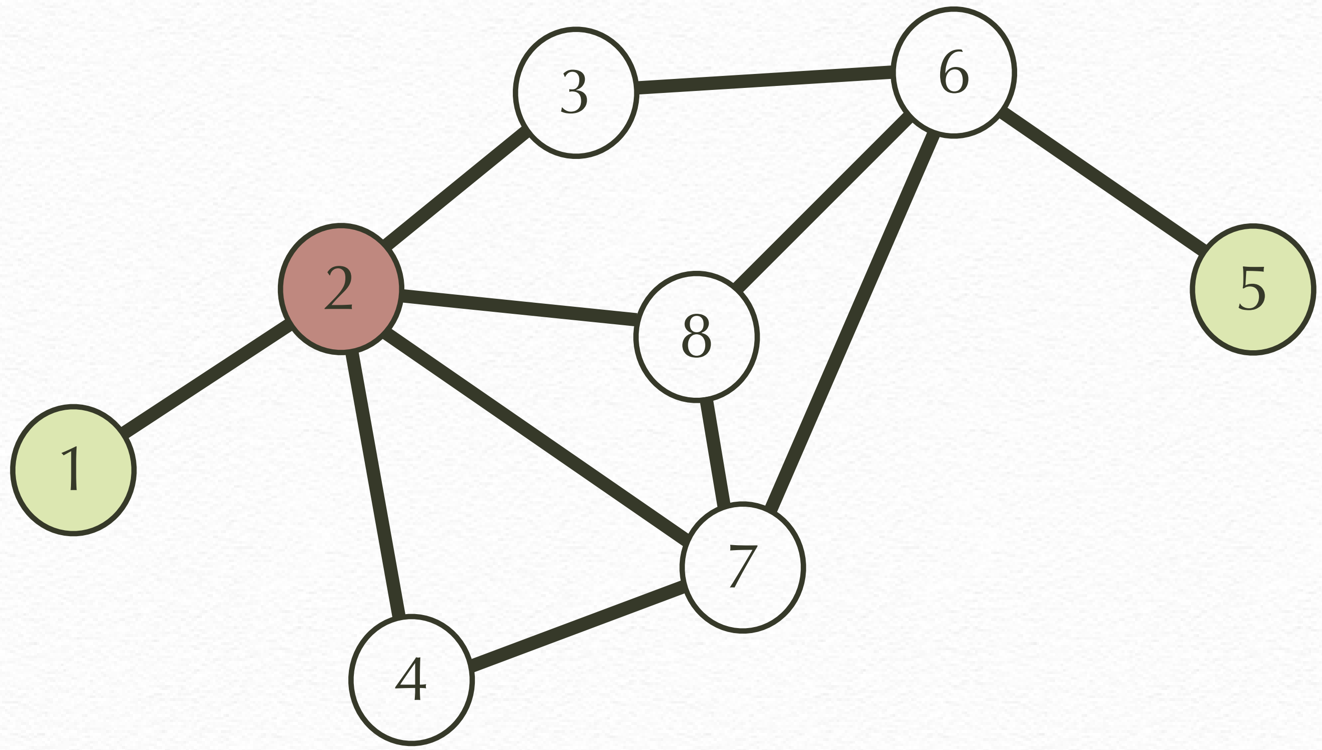


BulkRetry in LVish

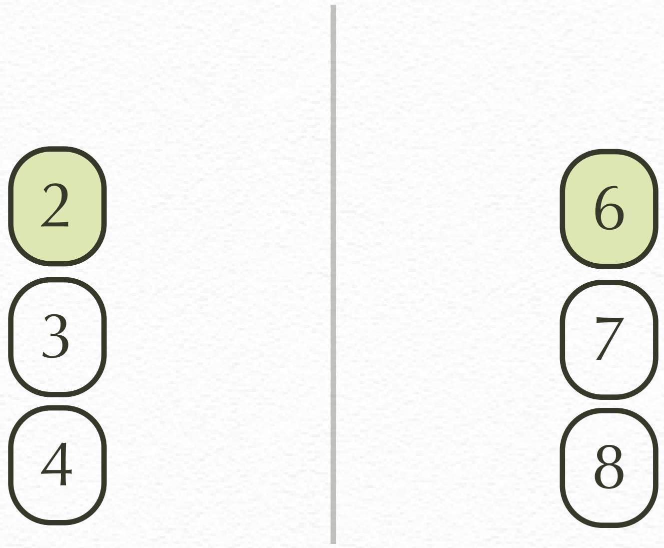




BulkRetry in LVish

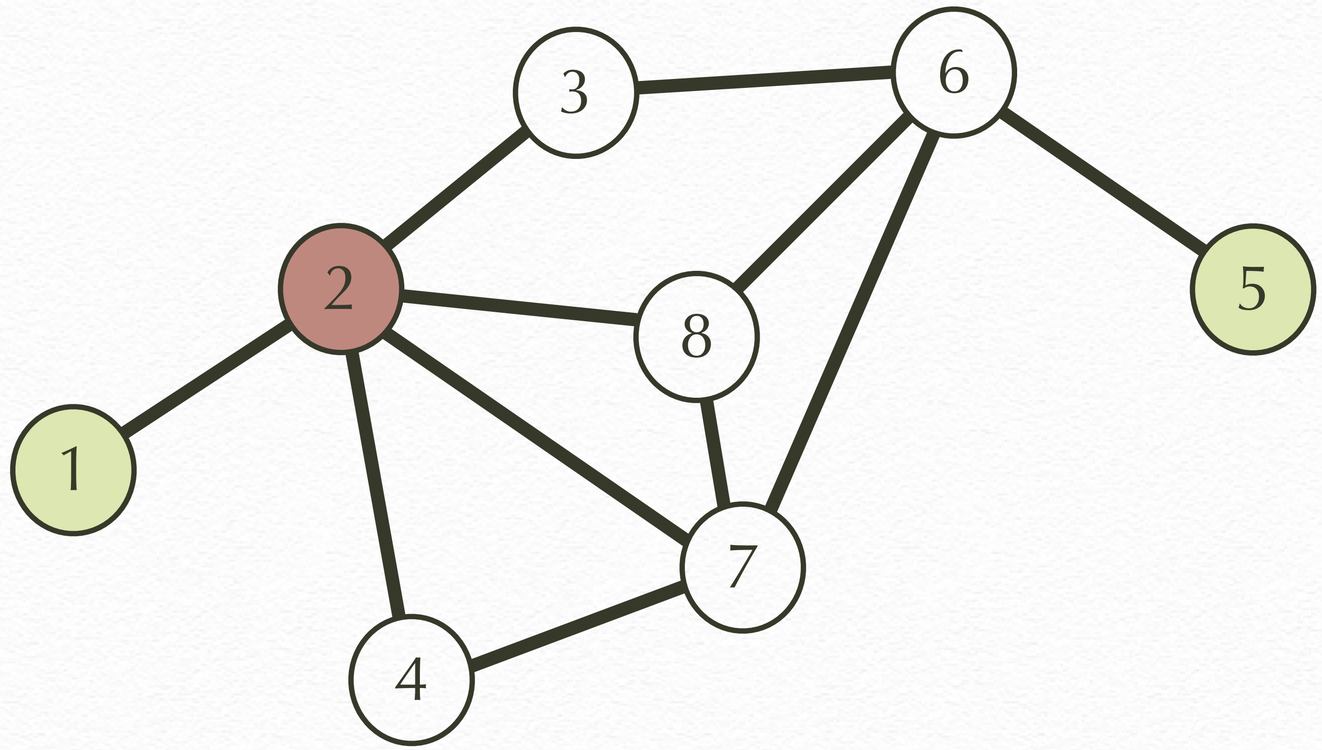


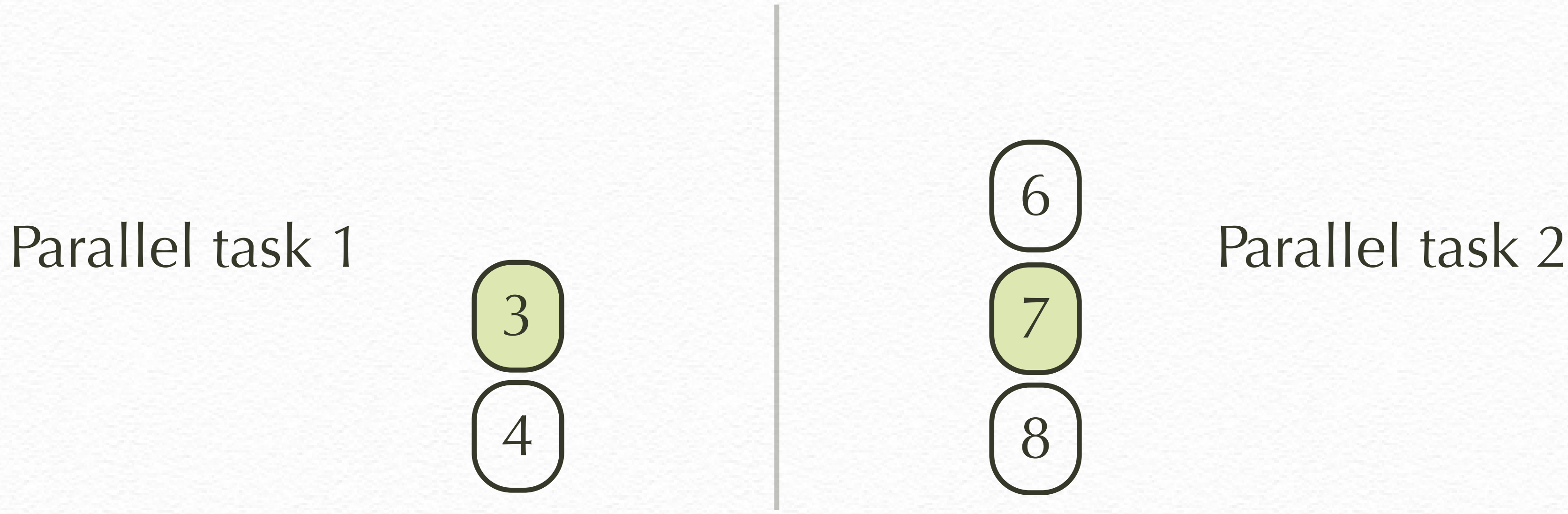
Parallel task 1



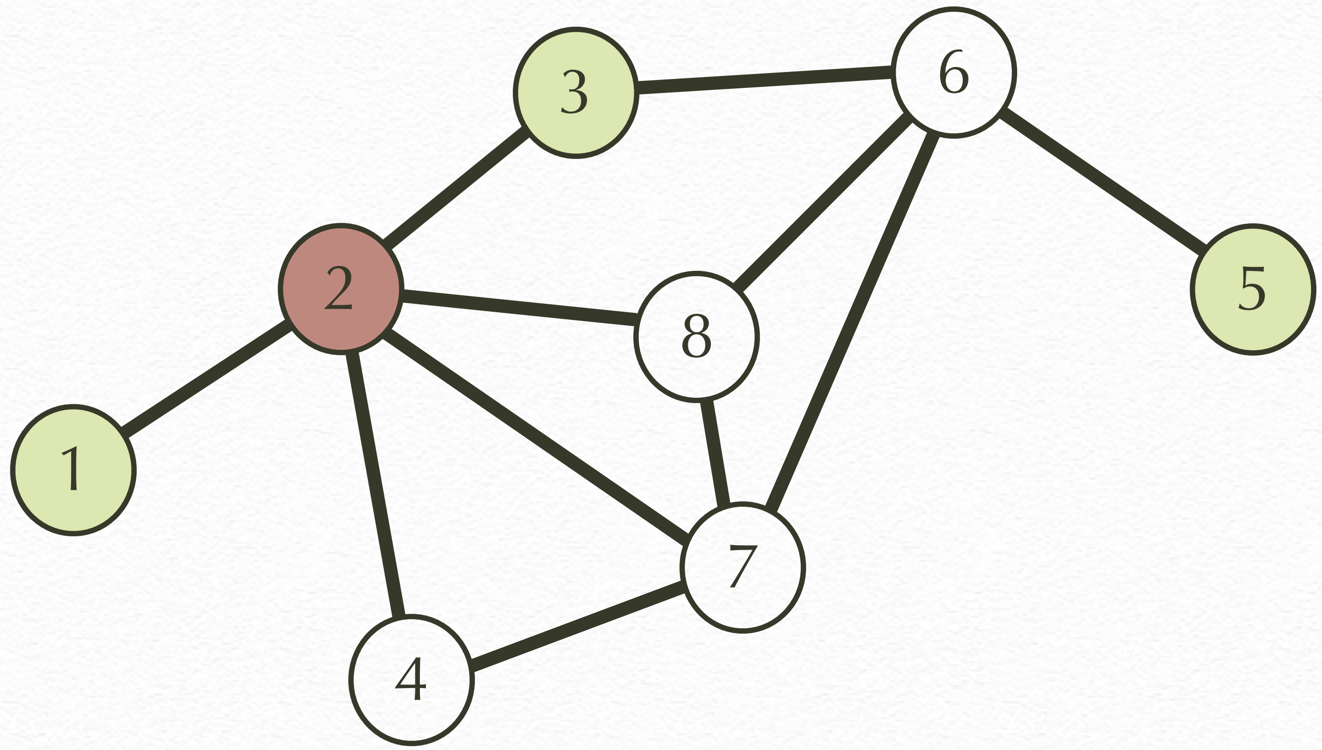
Parallel task 2

BulkRetry in LVish

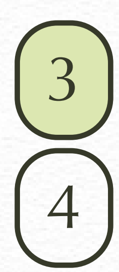


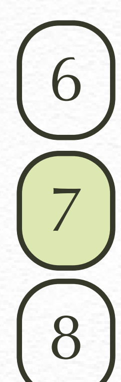


BulkRetry in LVish



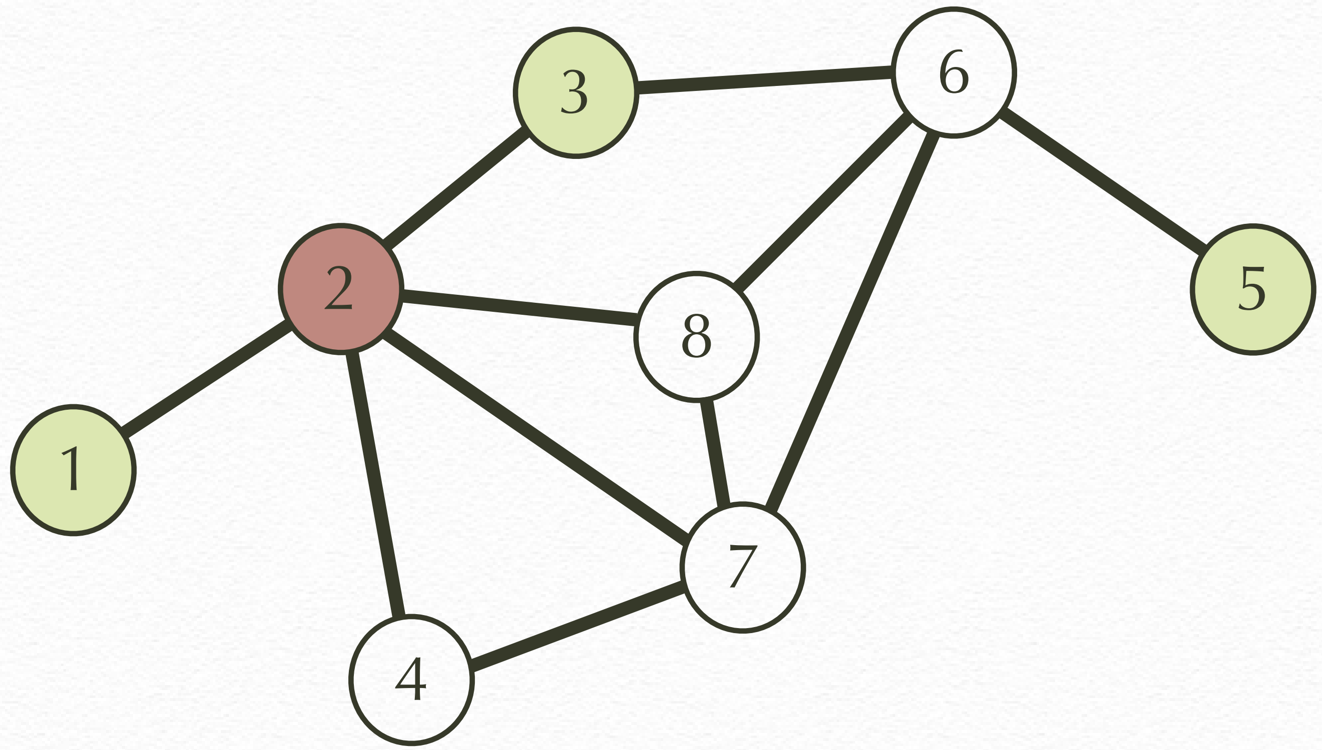
Parallel task 1



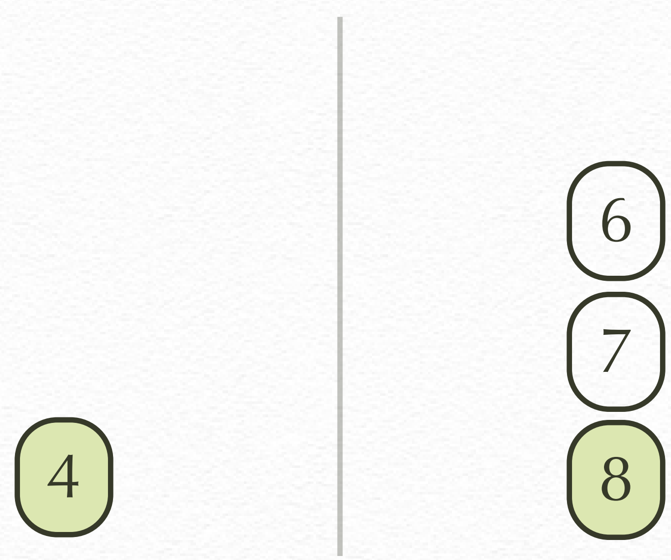


Parallel task 2

BulkRetry in LVish

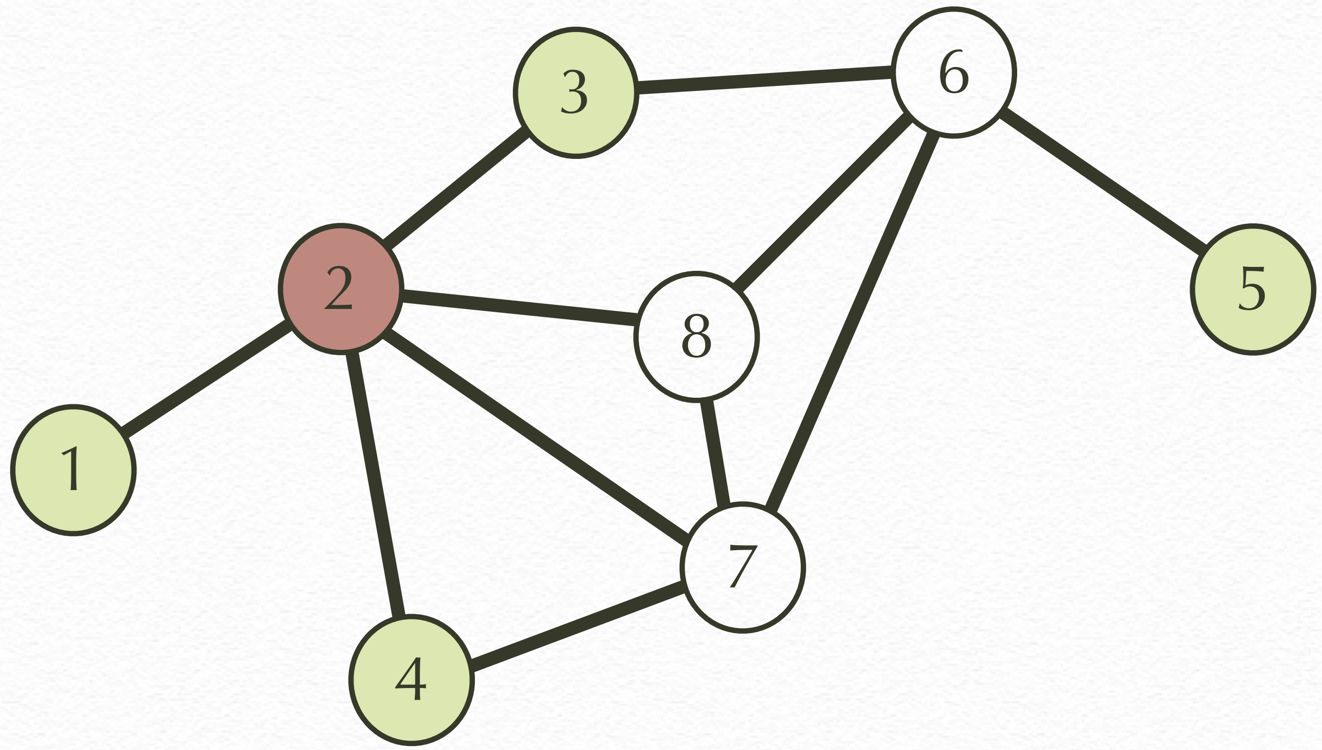


Parallel task 1

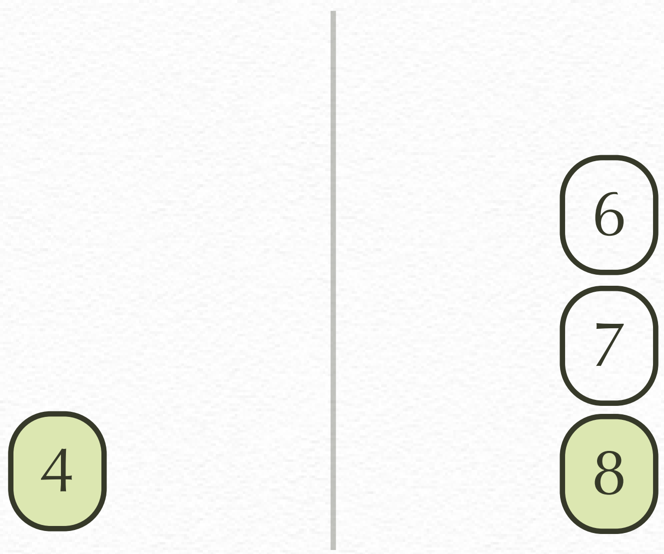


Parallel task 2

BulkRetry in LVish

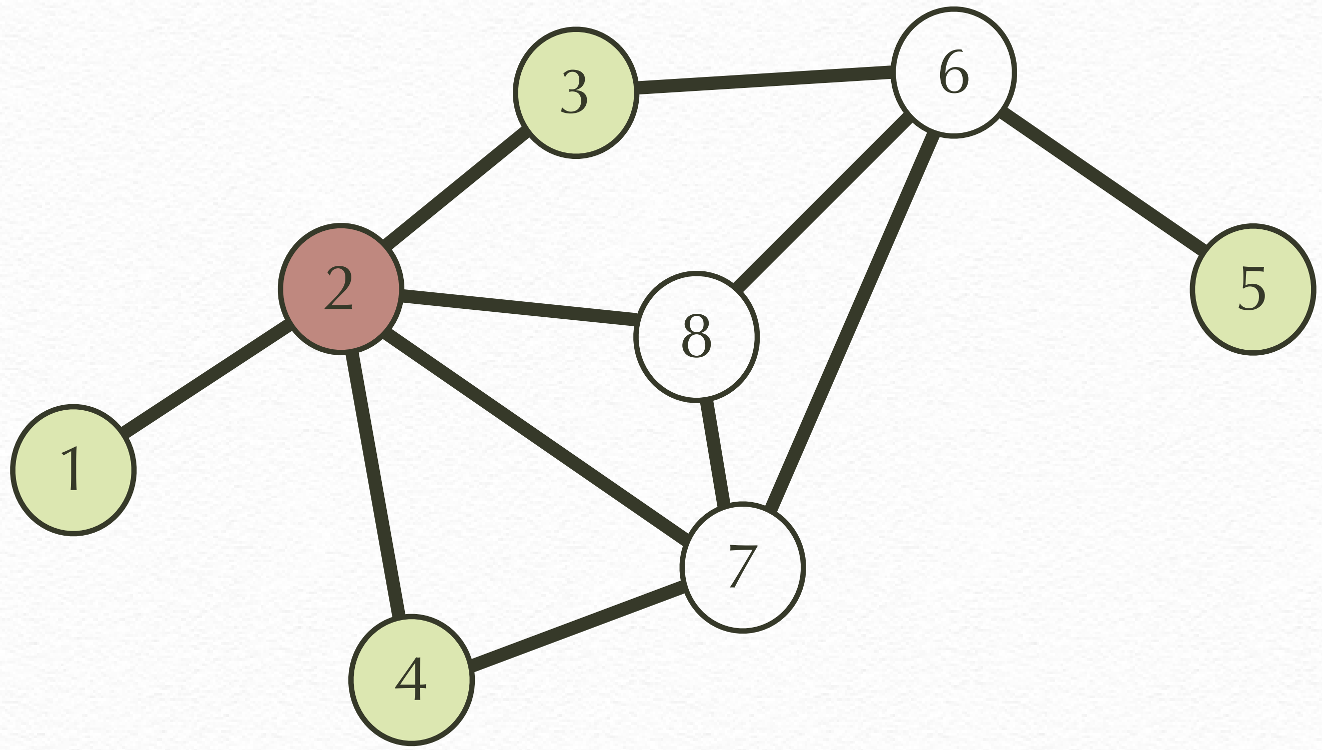


Parallel task 1



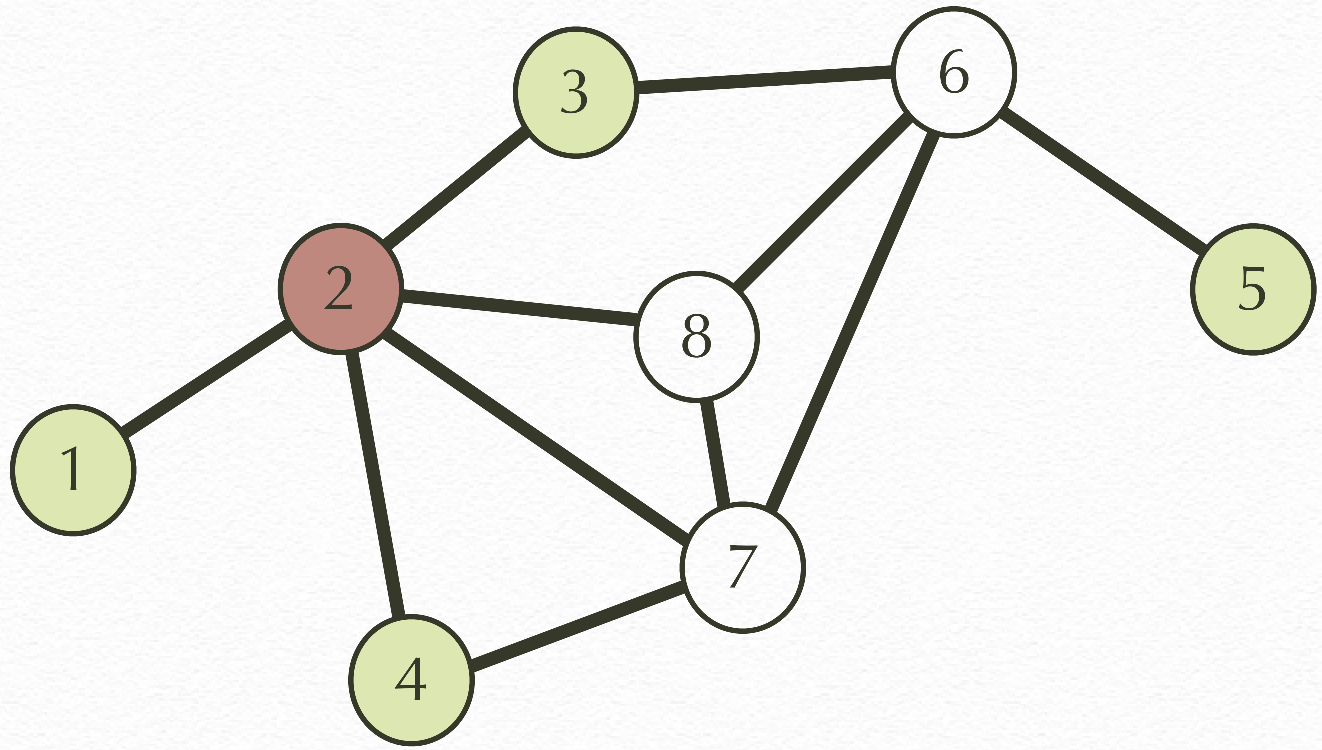
Parallel task 2

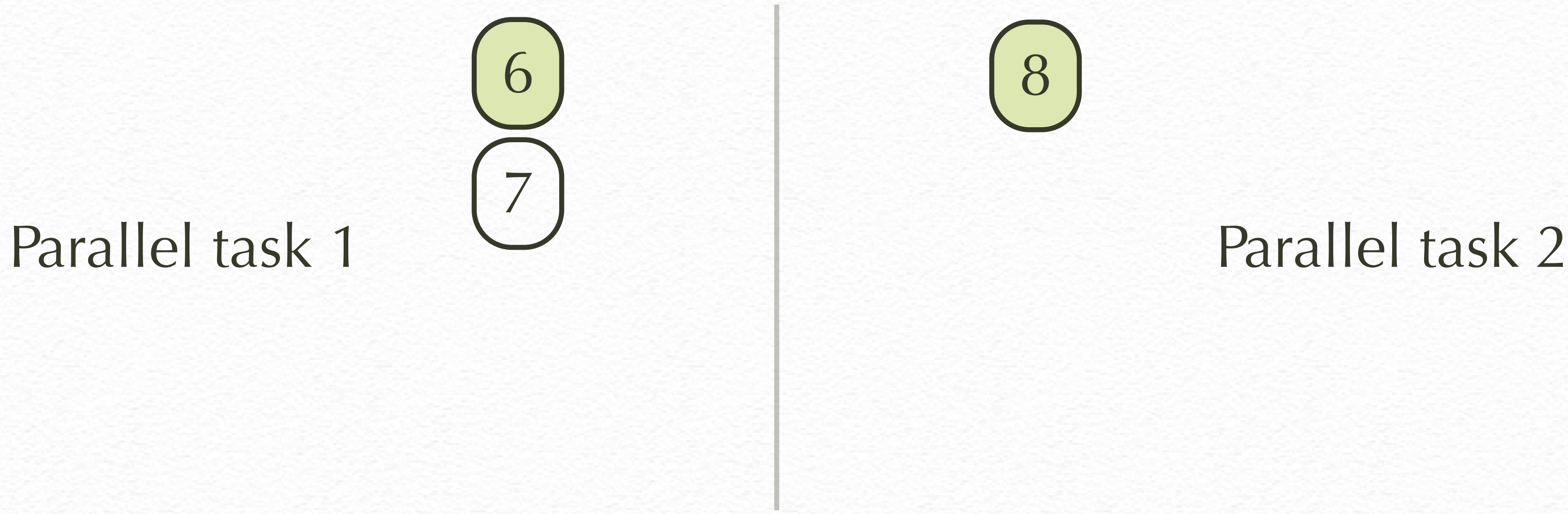
BulkRetry in LVish



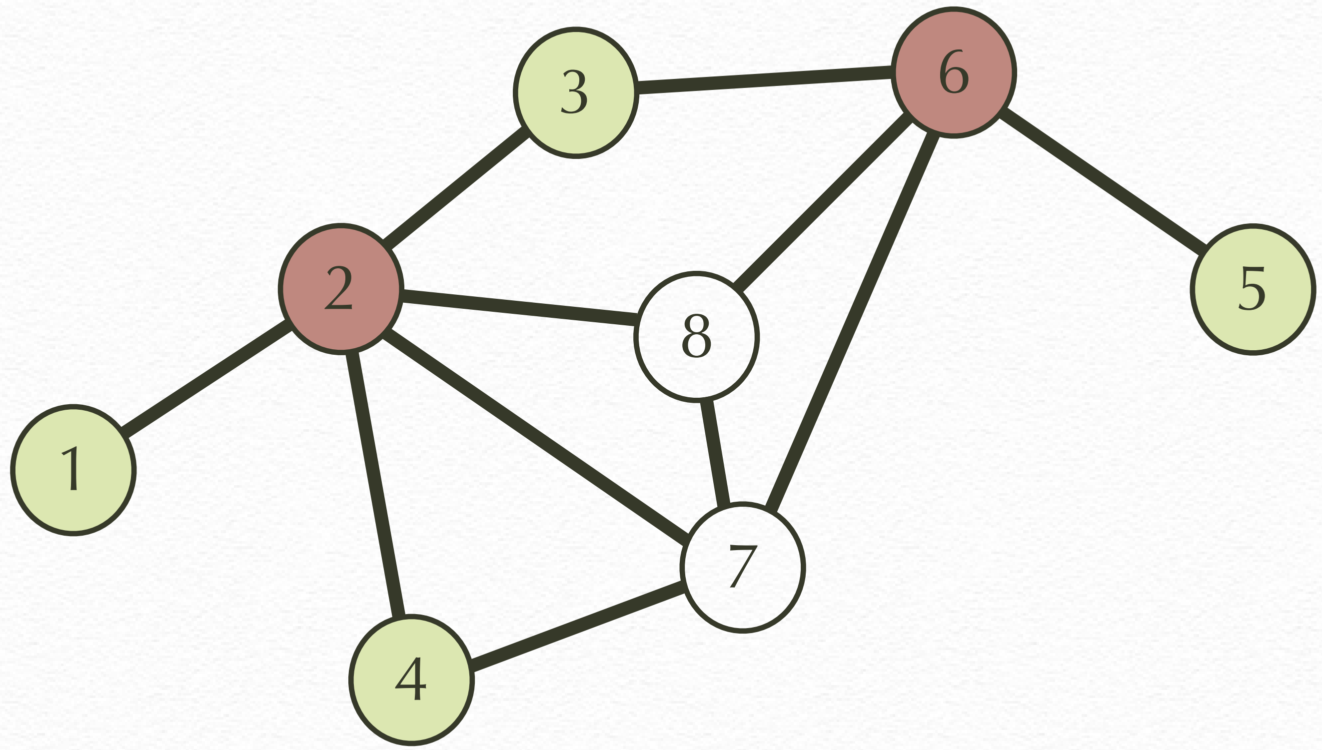
Parallel task 2

BulkRetry in LVish





BulkRetry in LVish



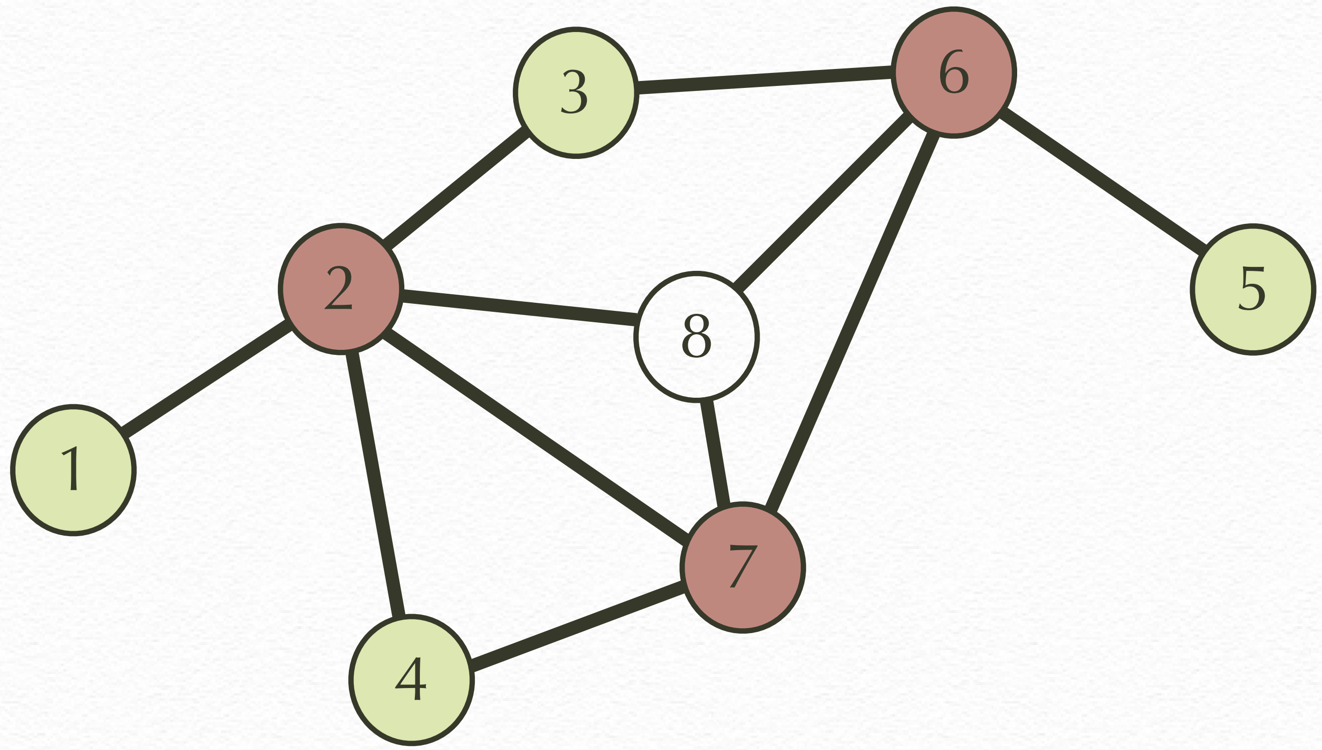
8

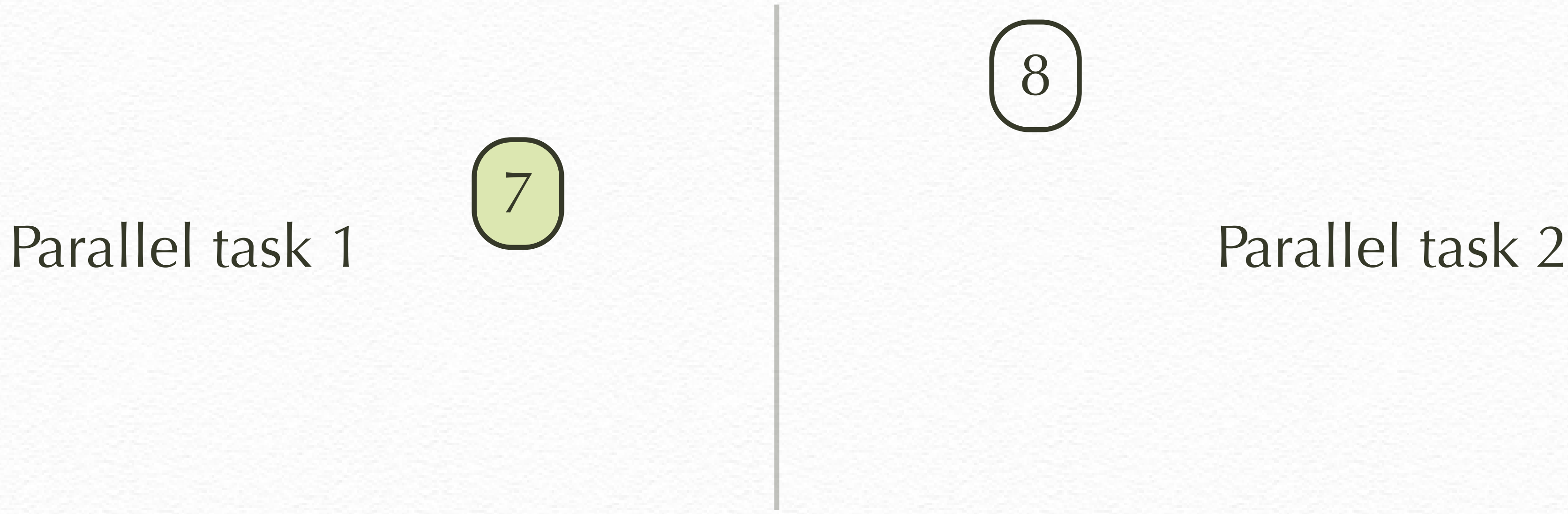
7

Parallel task 1

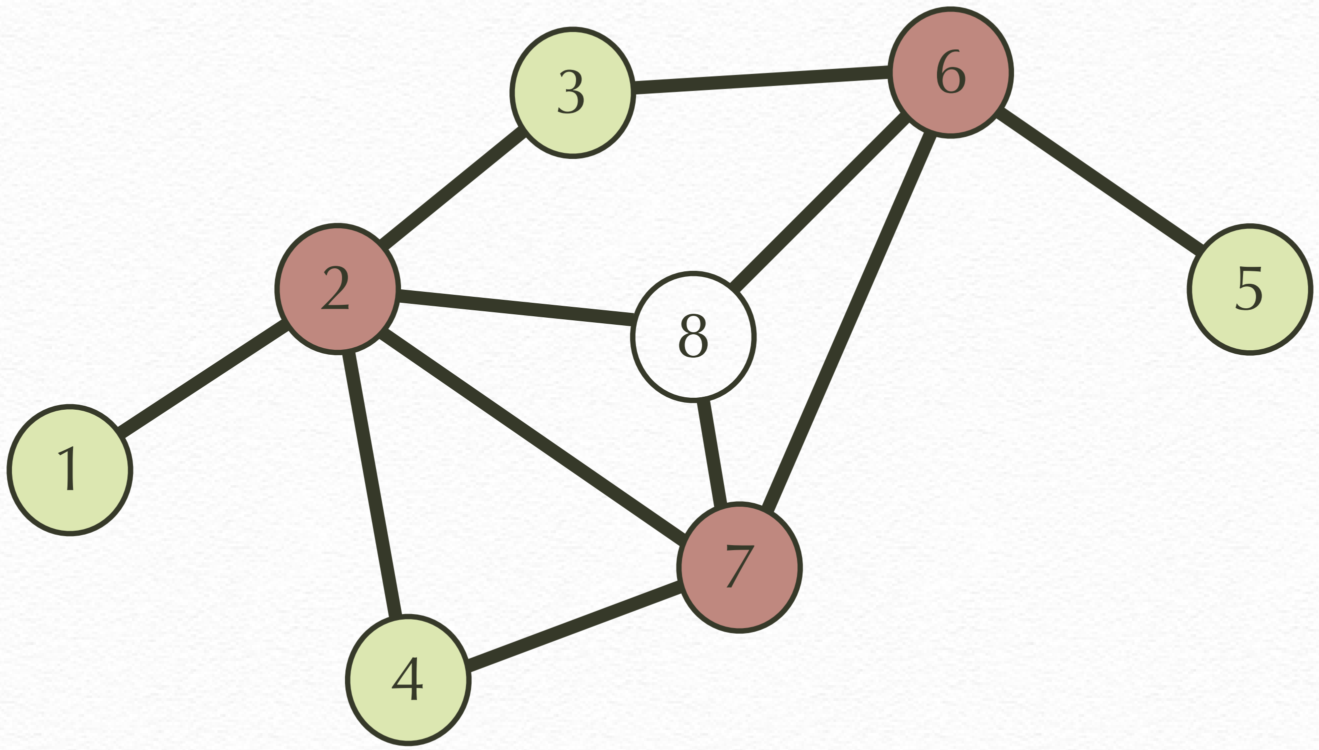
Parallel task 2

BulkRetry in LVish





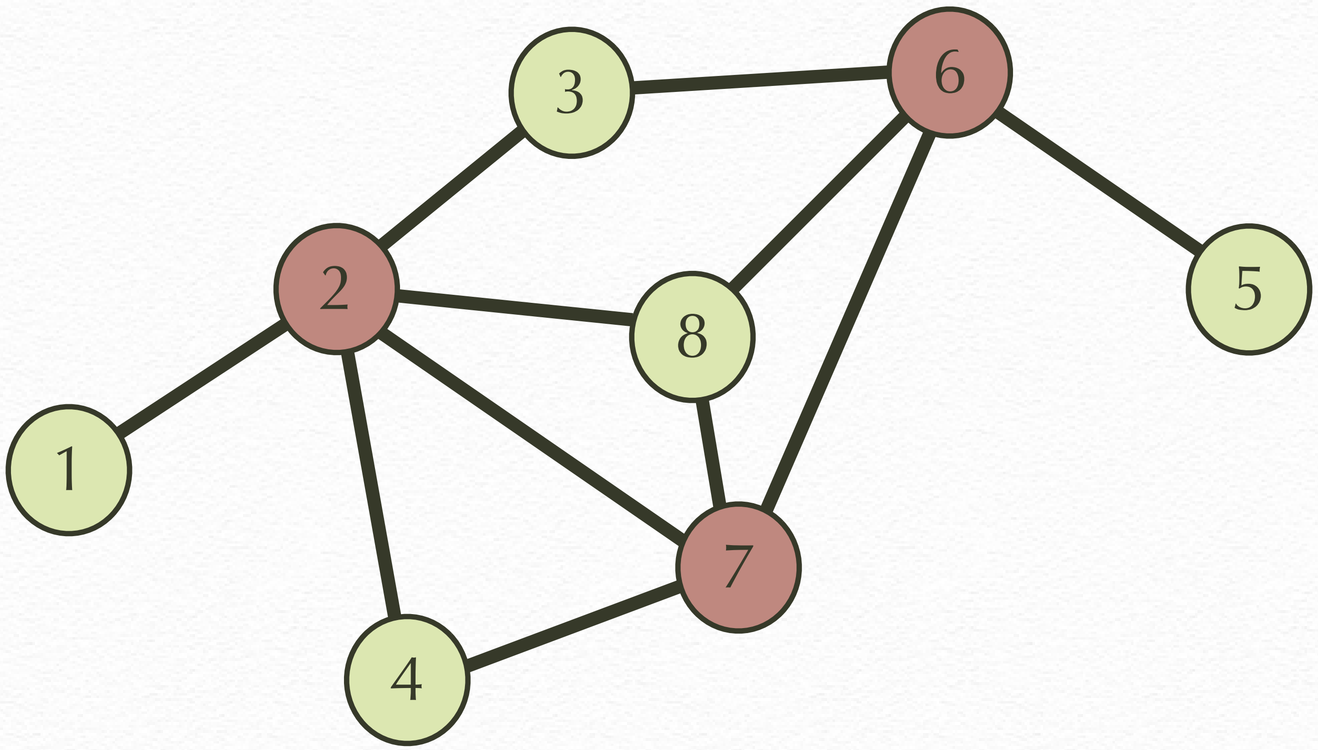
BulkRetry in LVish



8

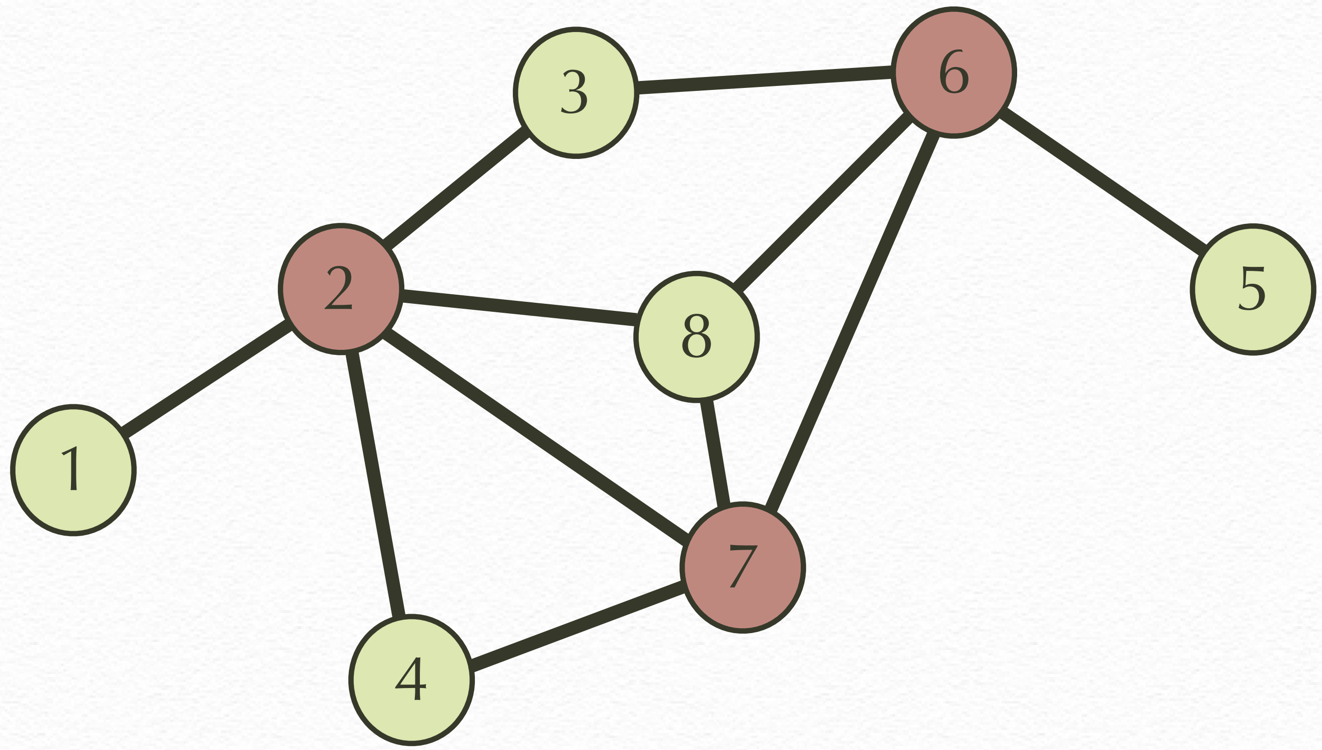
Parallel task 2

BulkRetry in LVish



Parallel task 2

BulkRetry in LVish



Evaluation

- Consistent speedup compared to vanilla LVish MIS
- LVish MIS gains 3x - 10x speedup
- 10x slower than PBBS (C++) implementations

Future work

- Randomized scheduler
- Constant-factor optimizations
- Improved data structures

Conclusion

- Compared approaches that deterministically process graphs
- Thank you!