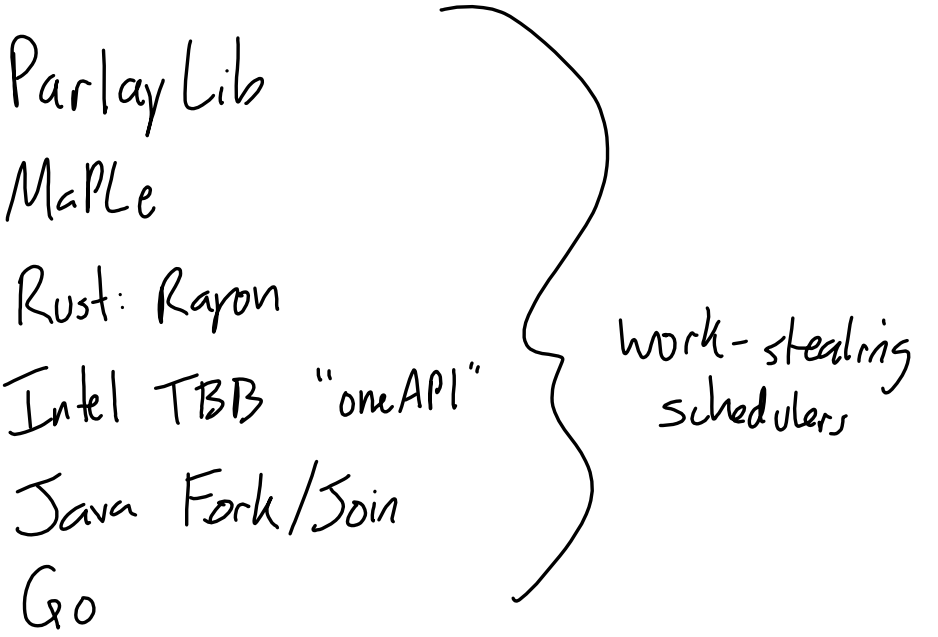
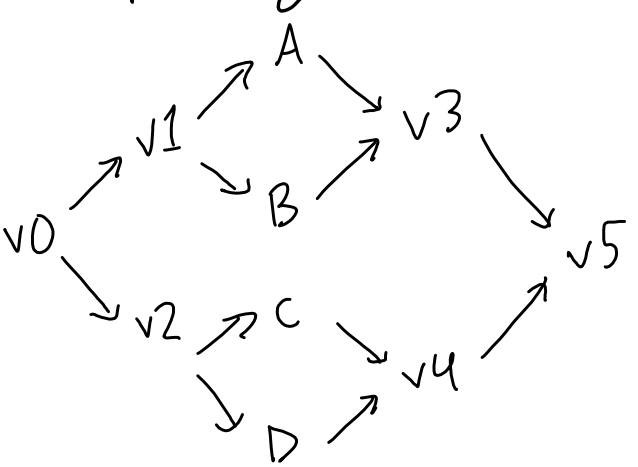

ParlayLib
MaPLe
Rust: Rayon
Intel TBB "oneAPI"
Java Fork/Join
Go
work-stealing schedulers

Greedy Scheduling

"offline" model

input: computation graph
(graph of tasks and dependencies)

output: assignment of processors to vertices
across time steps



2 processor schedule

t	P1	P2
1	v0	
2	v1	v2
3	A	C
4	B	D
5	v3	v4
6	v5	

greedy schedule:
if a task/vertex
is ready
and a processor is free,
then processor works
on that task

Greedy Scheduling is Close to Optimal

T = length of greedy schedule

Theorem:
for any greedy scheduler,

$$T \leq \frac{W}{P} + \frac{P-1}{P} S$$

$\frac{W}{P} \leq OPT$

$S \leq OPT$

$OPT \leq T$

$T \leq \frac{W}{P} + S$

$$\max\left(\frac{W}{P}, S\right) \leq OPT \leq T \leq 2 \cdot \max\left(\frac{W}{P}, S\right)$$

therefore: $T \leq 2 \cdot OPT$

Theorem:

for any greedy scheduler,

$$T \leq \frac{W}{P} + \frac{P-1}{P} S$$

Proof:

"work bucket" W "idle bucket" I

on each time step, if a processor is idle, put 1 token in the idle bucket.
Otherwise, 1 token in the work bucket.

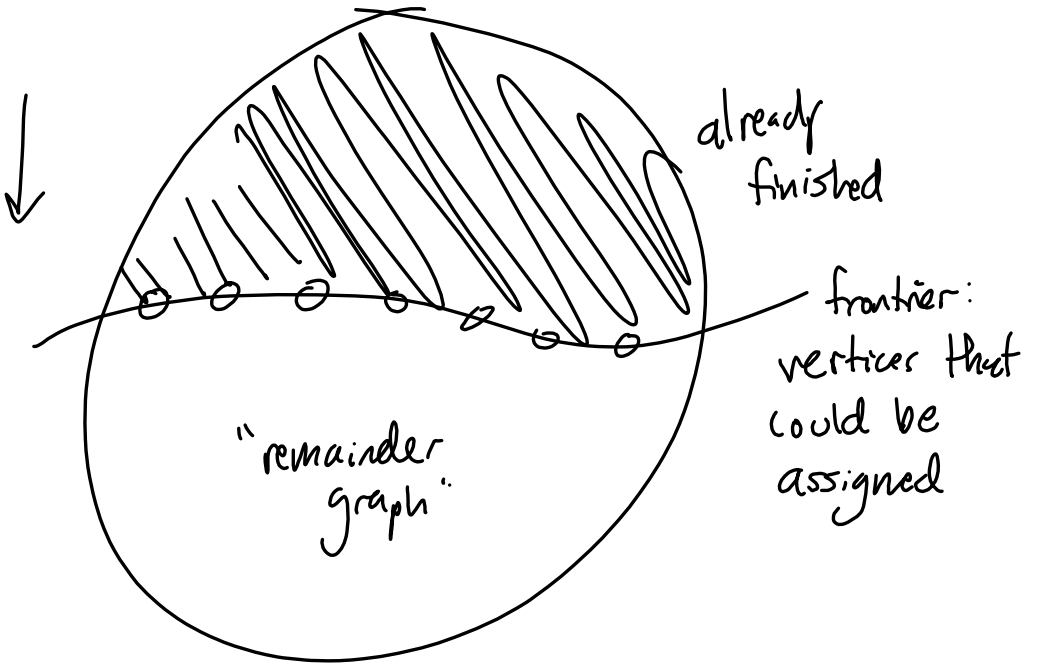
$$T \cdot P = W + I$$

---

two kinds of rounds:

saturated: exactly P work tokens

unsaturated: at least one idle token



unsaturated round: # ready $\leq P-1$

On unsaturated rounds, every ready vertex is assigned

Therefore, EVERY path in the remainder graph gets shorter

Therefore, span of the remainder graph decreases by at least 1

Therefore, # unsaturated rounds $\leq S$

Theorem:

for any greedy scheduler,

$$T \leq \frac{W}{P} + \frac{P-1}{P} S$$

Proof:

"work bucket" W "idle bucket" I

on each time step, if a processor is idle, put 1 token in the idle bucket. Otherwise, 1 token in the work bucket.

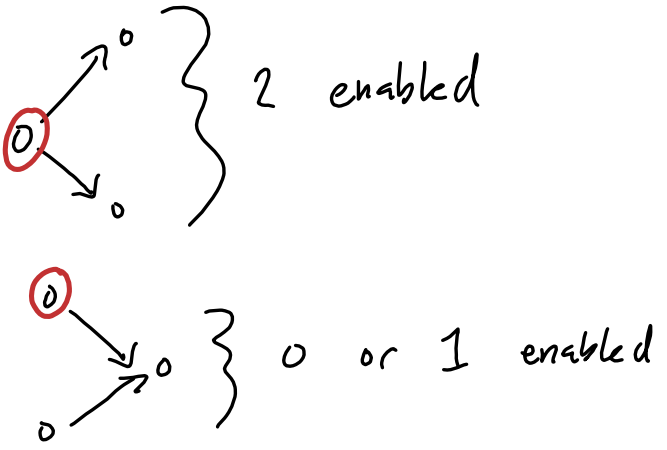
$$T \cdot P = W + I$$

$$I \leq (\#\text{unsaturated}) \cdot (P-1) \leq S(P-1)$$

$$TP \leq W + S(P-1)$$

$$T \leq W/P + S\frac{P-1}{P}$$

Attempt: Online Greedy Scheduling

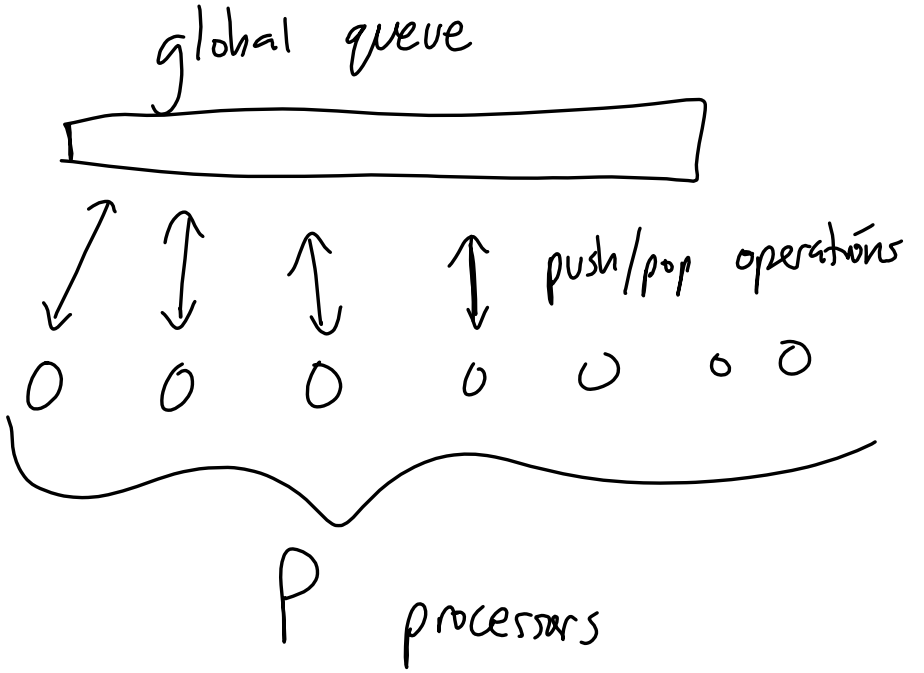


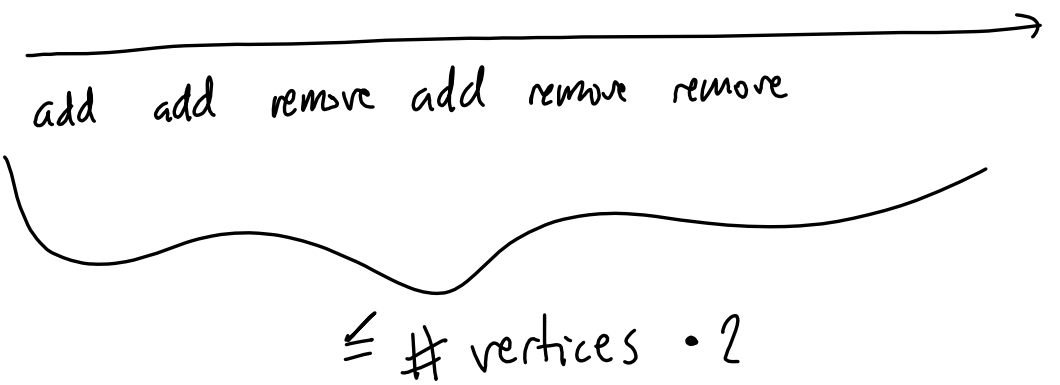
execute(vertex) → list of enabled vertices
(either 0, 1, or 2)

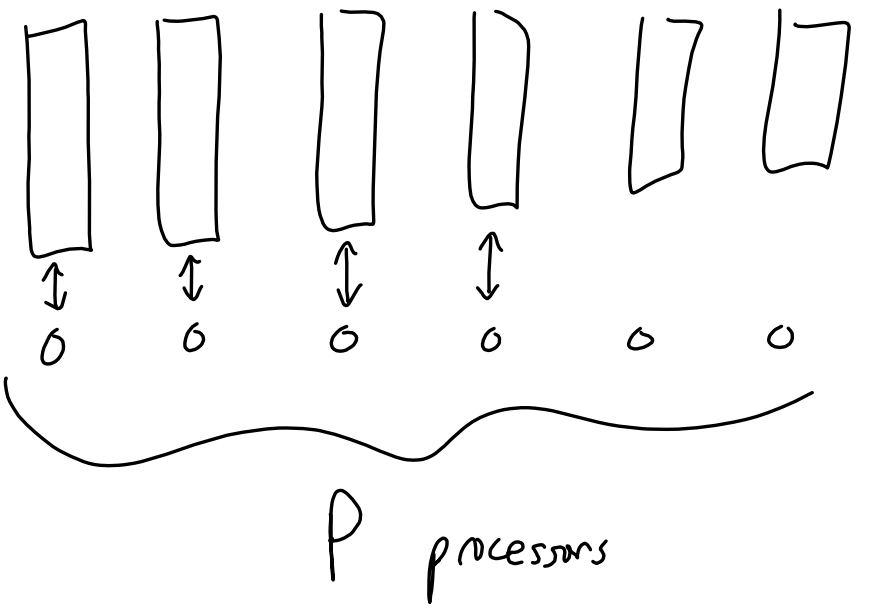
case 1: assign to self

case 0: try to take from global queue

case 2: assign one to self,
put the other in the queue


global queue
push/pop operations
P processors


add add remove add remove remove
≤ # vertices • 2



"steal": take a vertex from another processor

round-robin stealing: walk left-to-right, looking for a vertex to steal

"symmetry breaking": how to get identical parallel processes to <u>not</u> align

Work Stealing

if idle:

pick a random victim,
try to steal
if successful: start working
otherwise: continue idle (continue stealing)

if working:

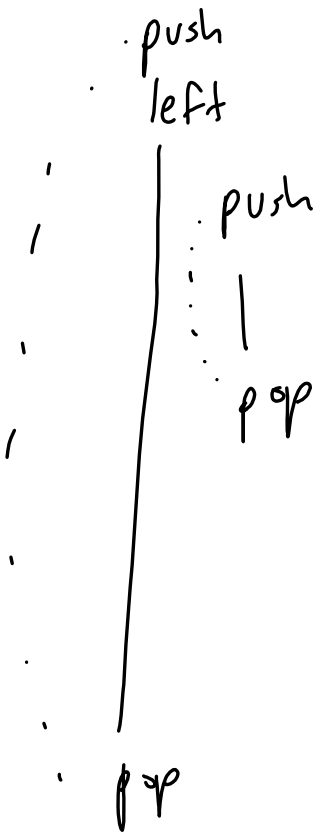
work on local queue as much as possible, until it is empty (and then start stealing)

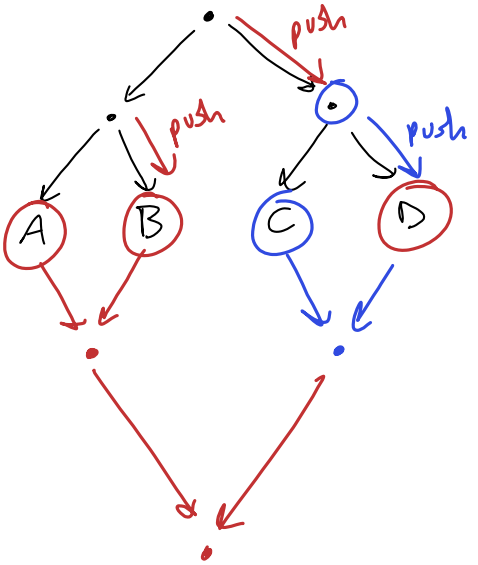
Steal from the "top"

local push/pop from the "bottom"

doubly-ended queues "deques"

"Balls in Bins"


push
left
push
pop
pop



"burying the join point"