Cooperative Kernels: GPU Multitasking for Blocking Algorithms

Tyler Sorensen, Hugues Evrard, Alastair Donaldson
Imperial College London, UK

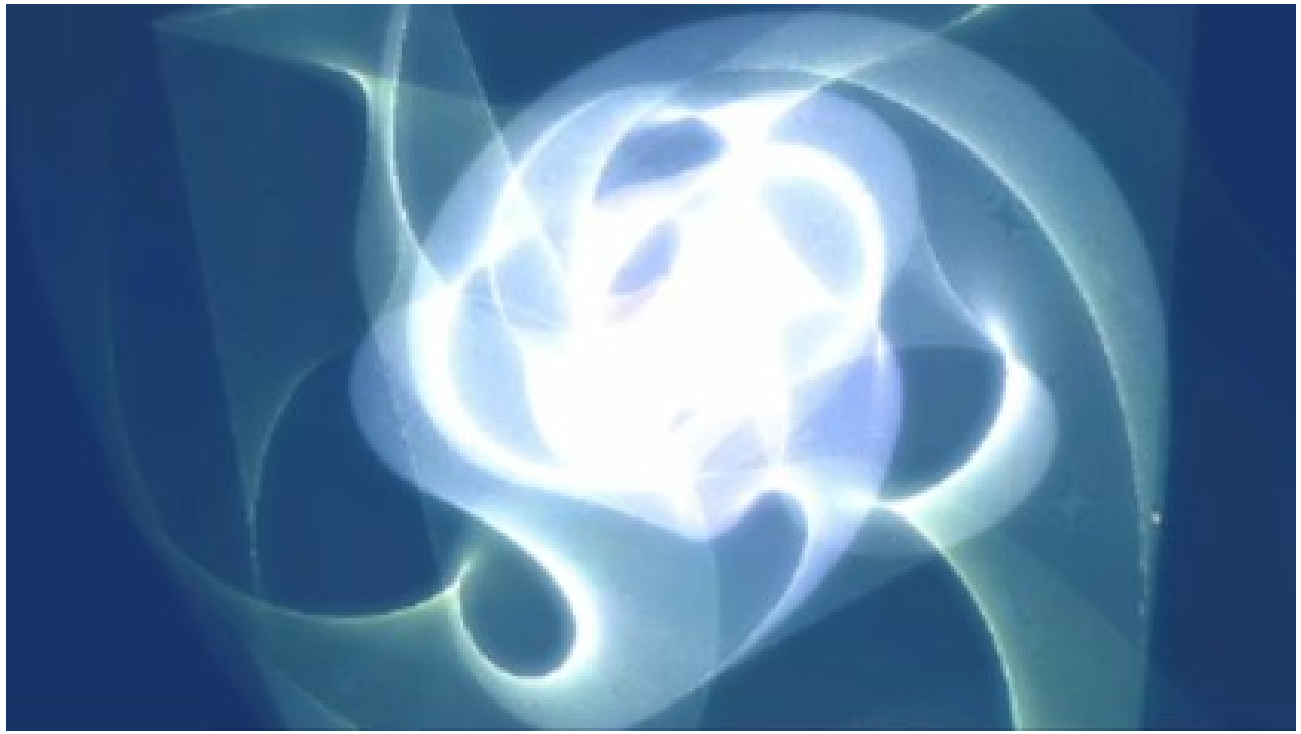
FSE
Sept. 2017

“The GPU Multitasking Talk”

Tyler Sorensen, Hugues Evrard, Alastair Donaldson
Imperial College London, UK

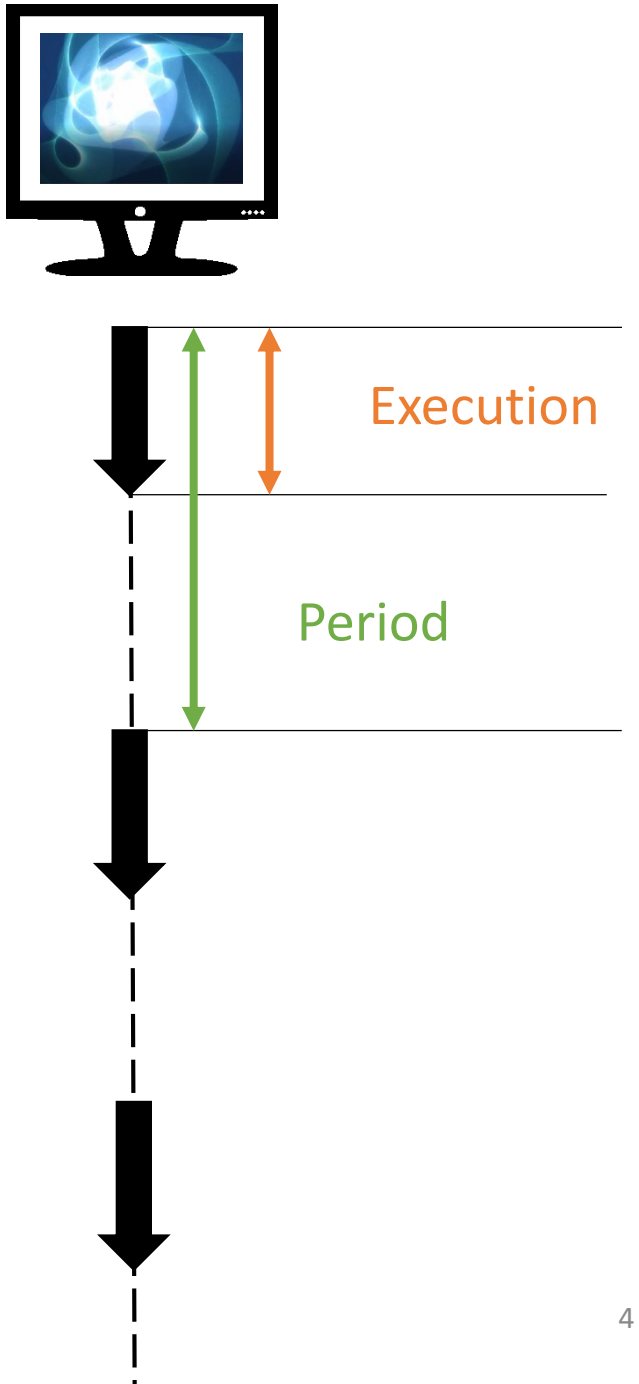
FSE
Sept. 2017

Graphics execution



from http://webglsamples.org/electricflower/electricflower.html

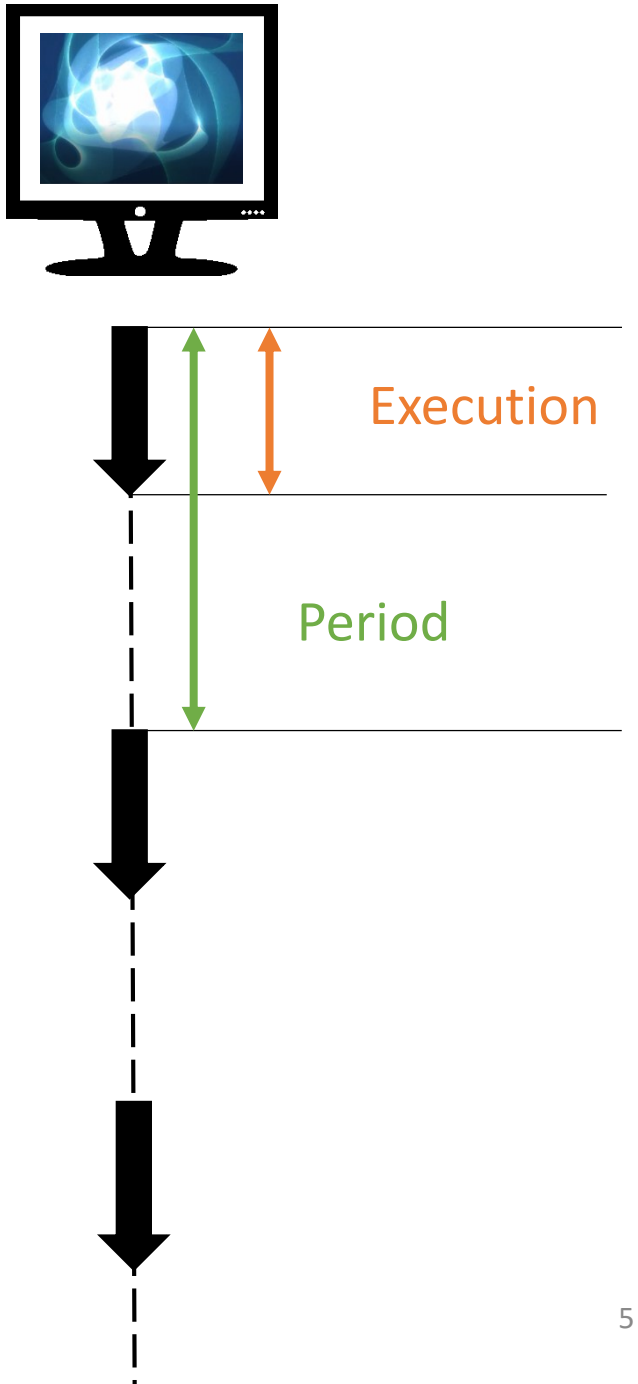
Graphics execution



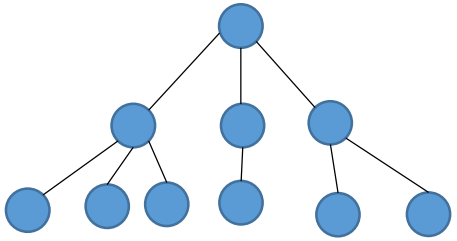
from http://webglsamples.org/electricflower/electricflower.html

Graphics execution

Workload	Period (ms)	Execution (ms)
Light	70	3
Medium	40	3
Heavy	40	10



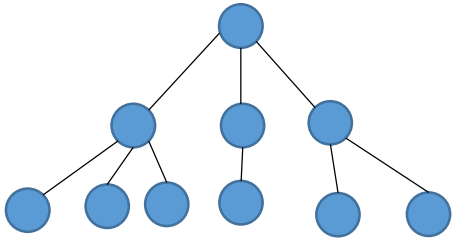
Compute execution (sssp)



Dataset	Execution (ms)
NY-road	400
CAL-road	1200
USA-road	2200

Compute GPU program == *kernel*

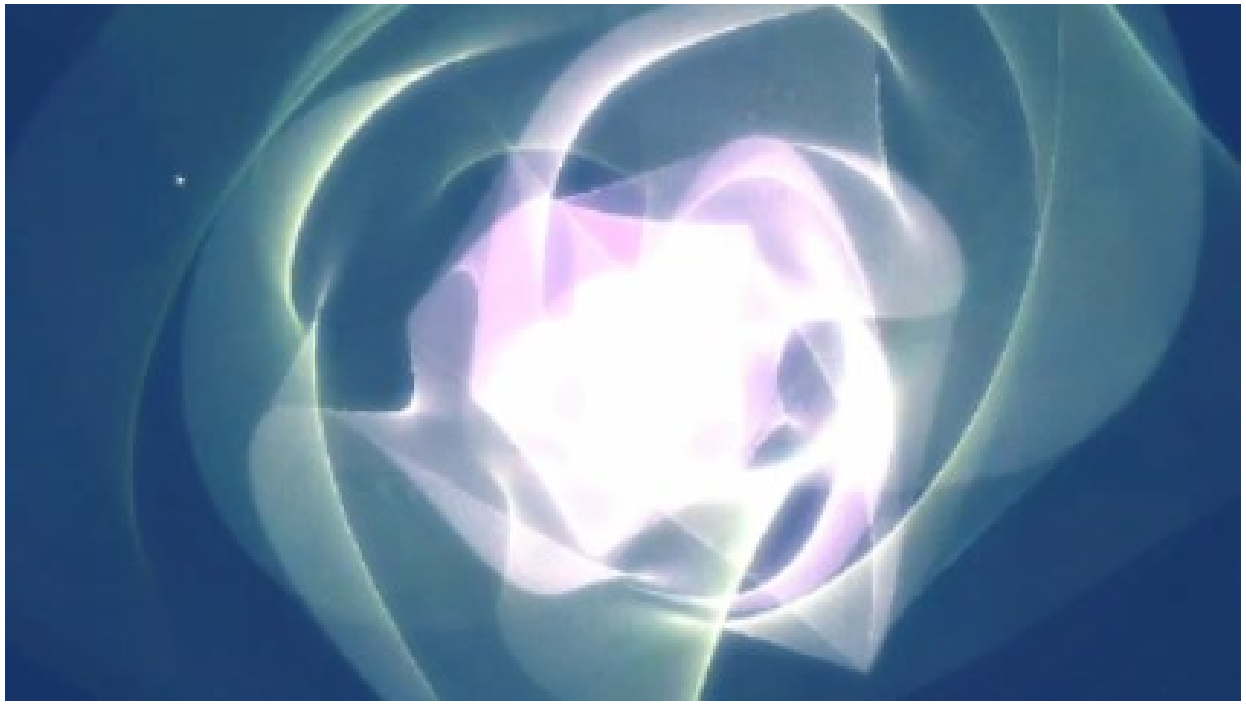
Compute execution (sssp)



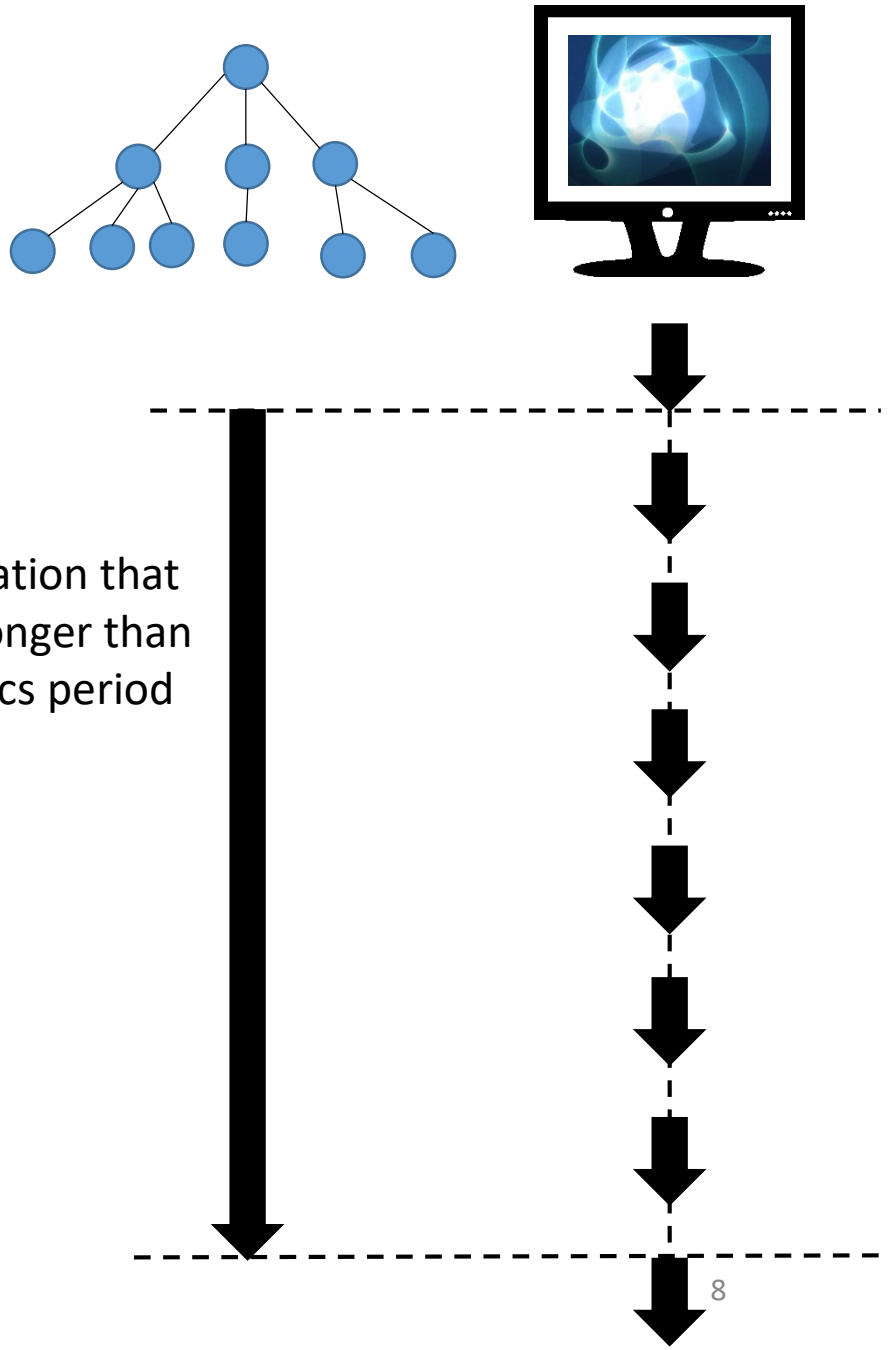
Dataset	Execution (ms)
NY-road	400
CAL-road	1200
USA-road	2200

Workload	Period (ms)	Execution (ms)
Light	70	3
Medium	40	3
Heavy	40	10

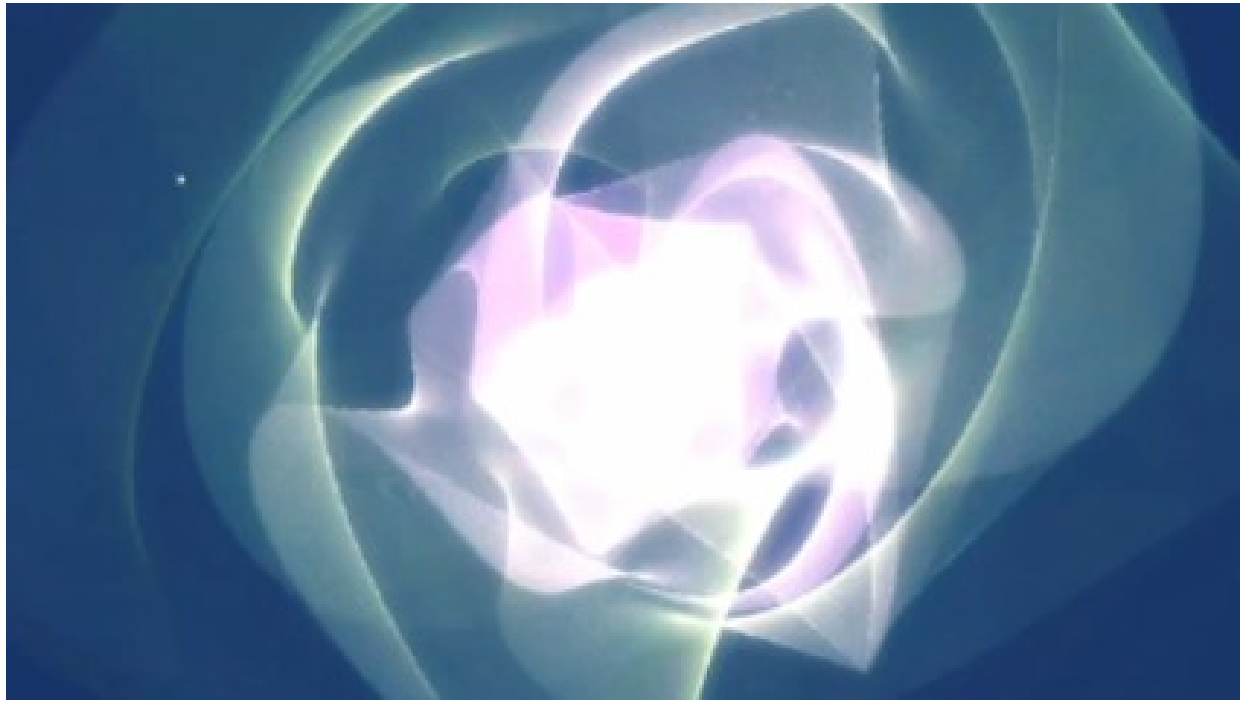
Multi-tasking



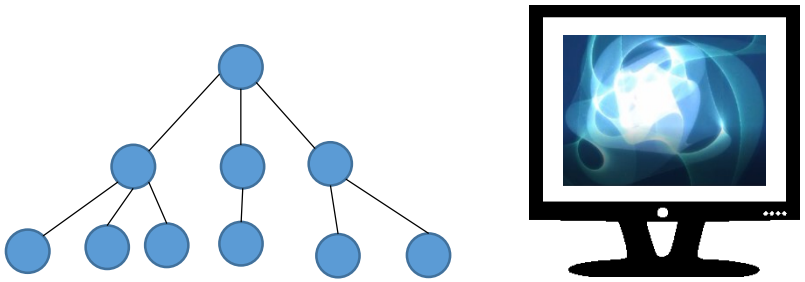
Running http://webglsamples.org/electricflower/electricflower.html
with sssp OpenCL application in background on Intel HD5500

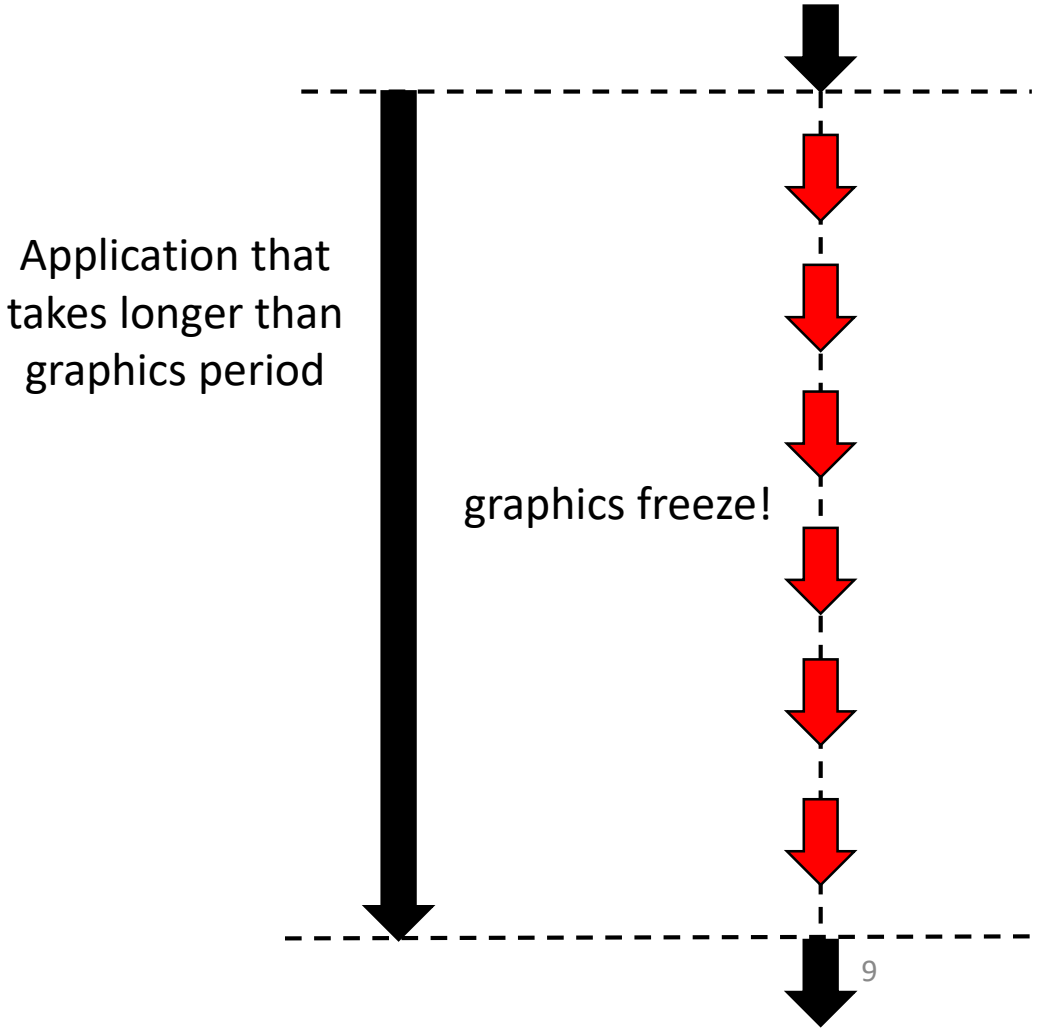


Multi-tasking

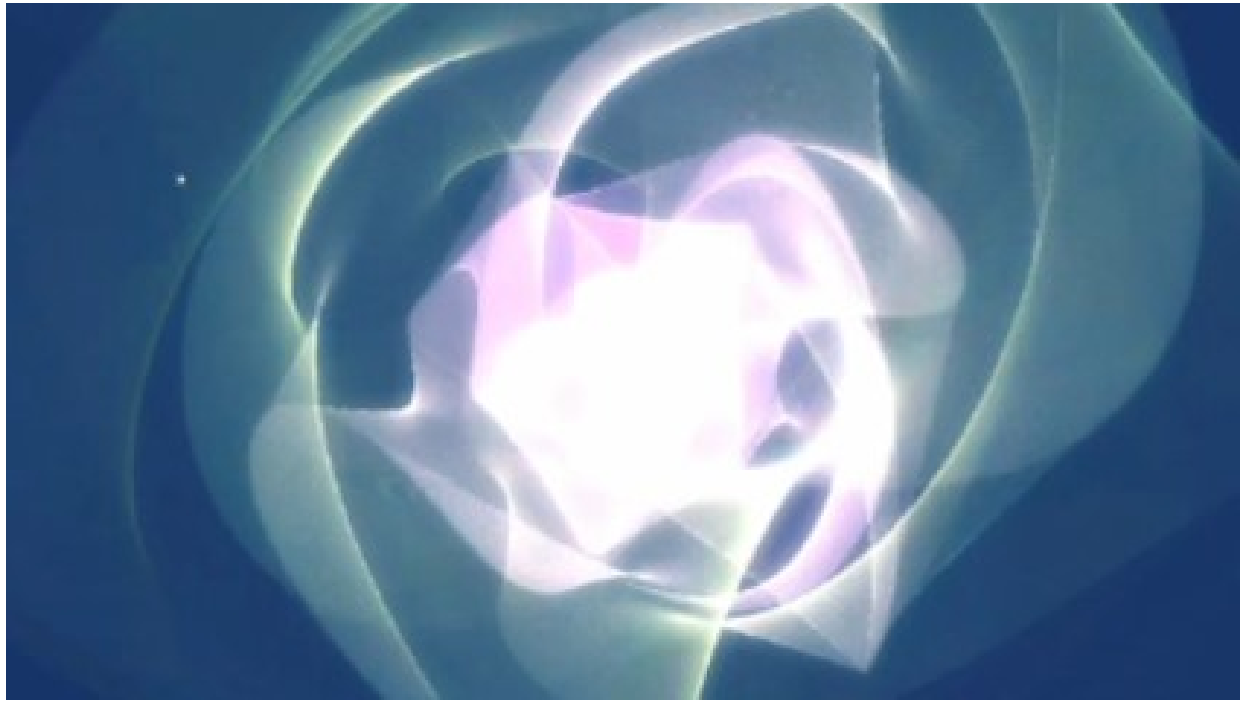


Running http://webglsamples.org/electricflower/electricflower.html
with sssp OpenCL application in background on Intel HD5500

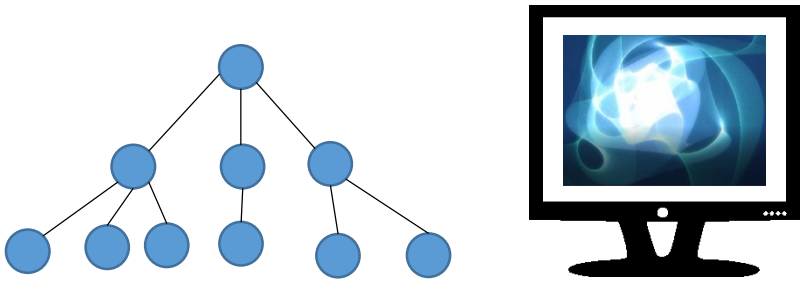


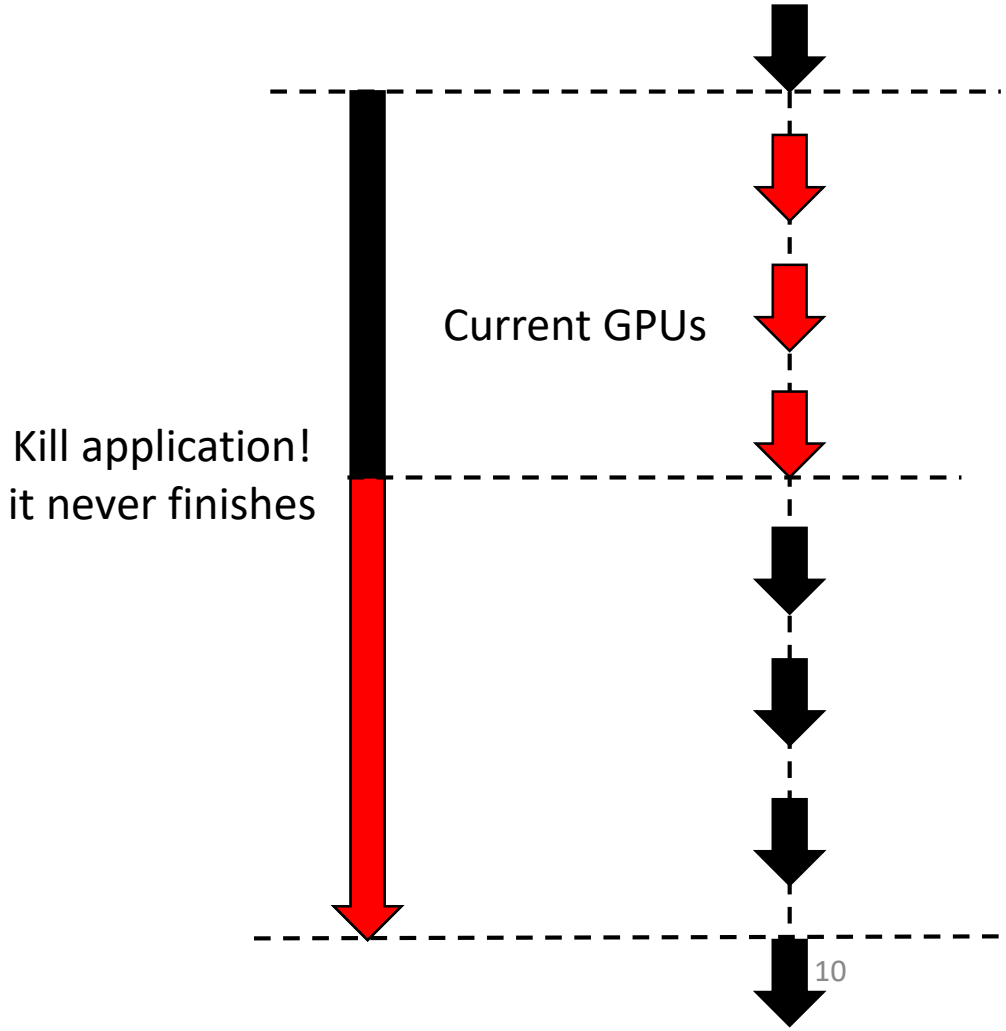


Multi-tasking

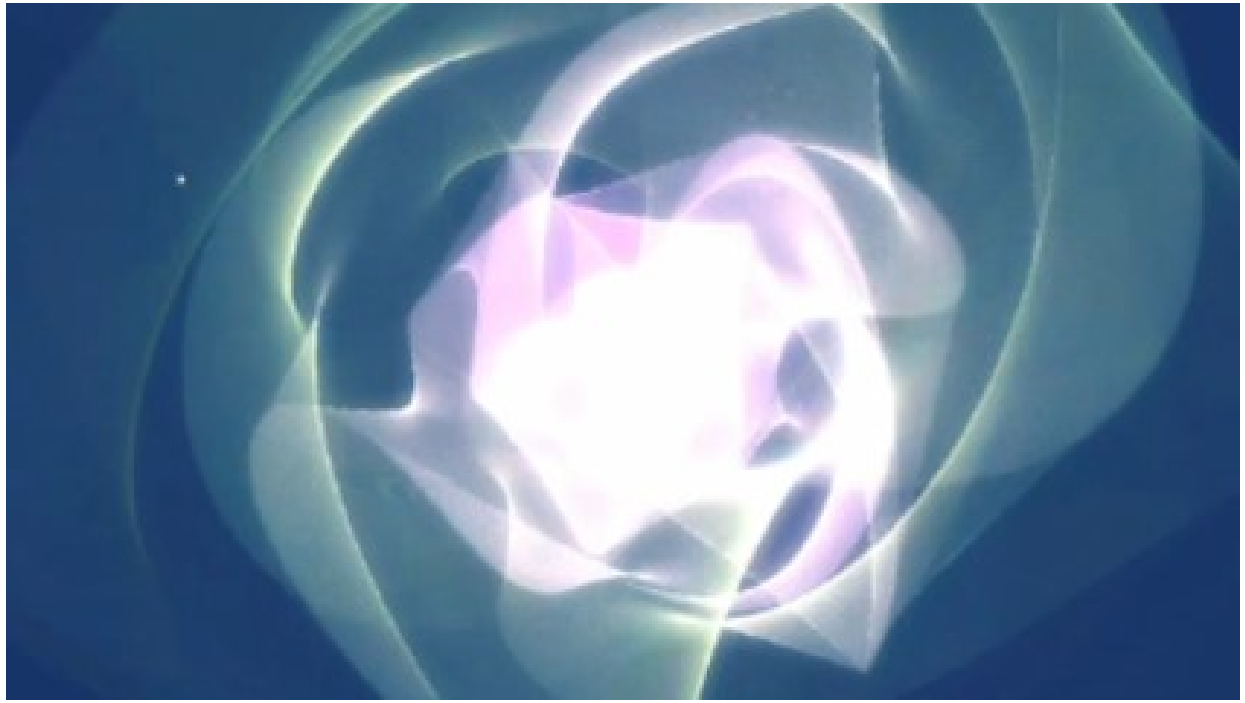


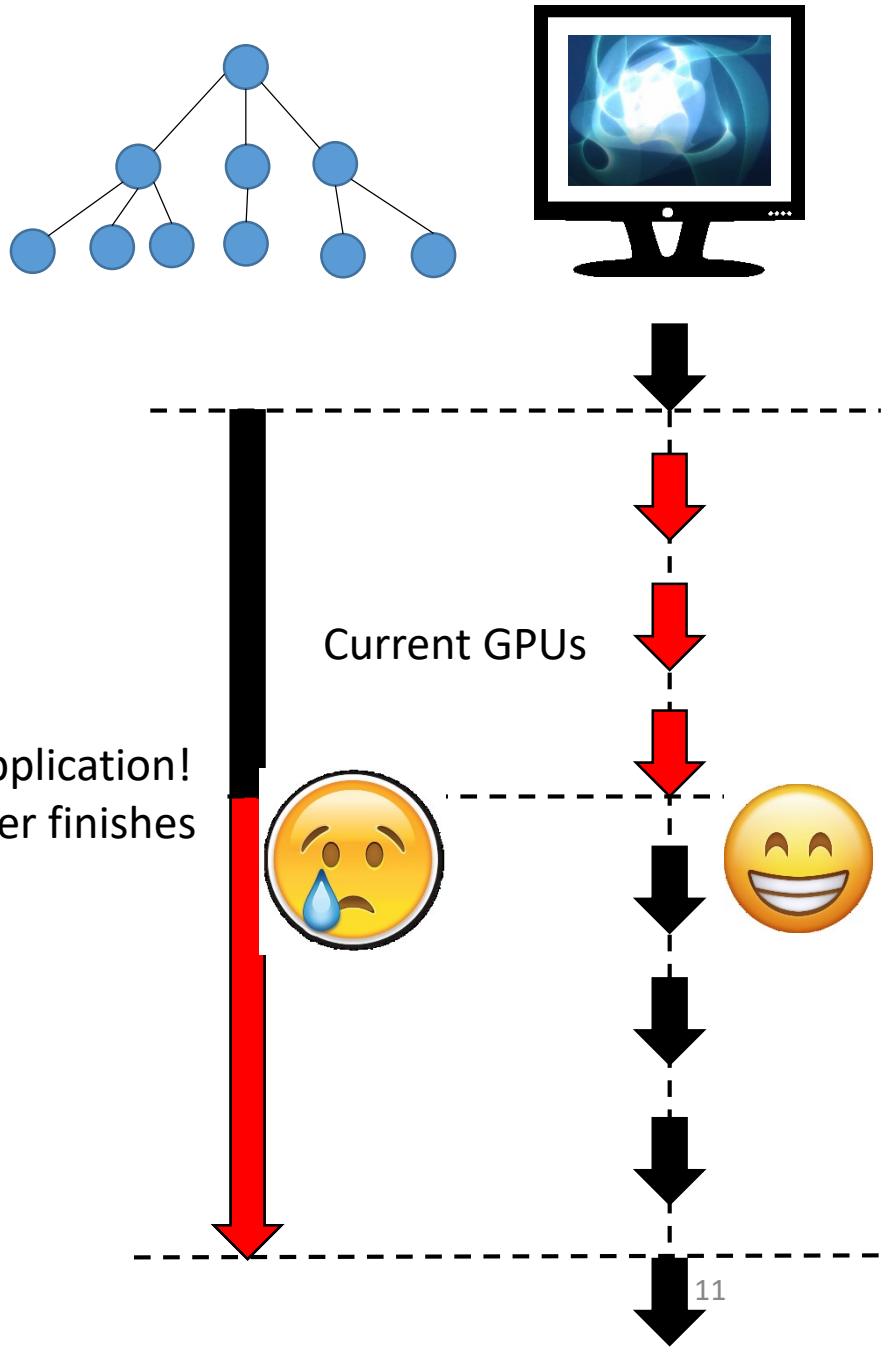
Running http://webglsamples.org/electricflower/electricflower.html
with sssp OpenCL application in background on Intel HD5500





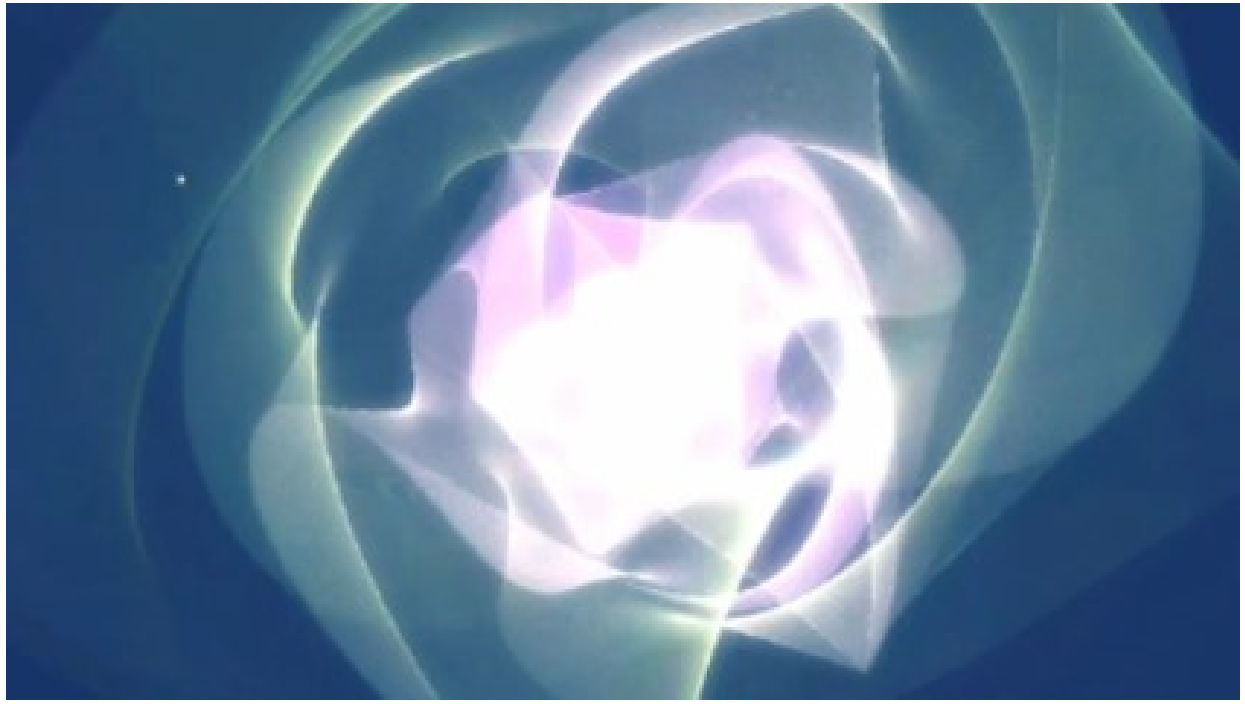
Multi-tasking



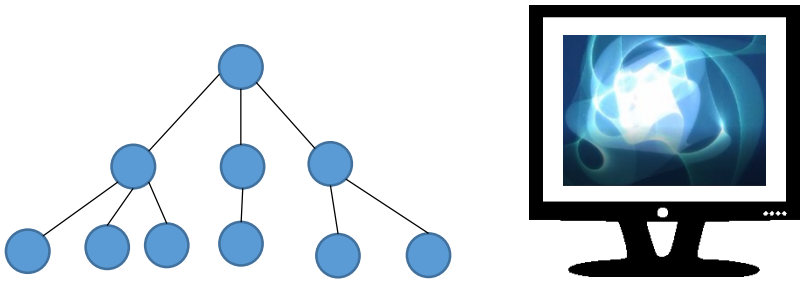


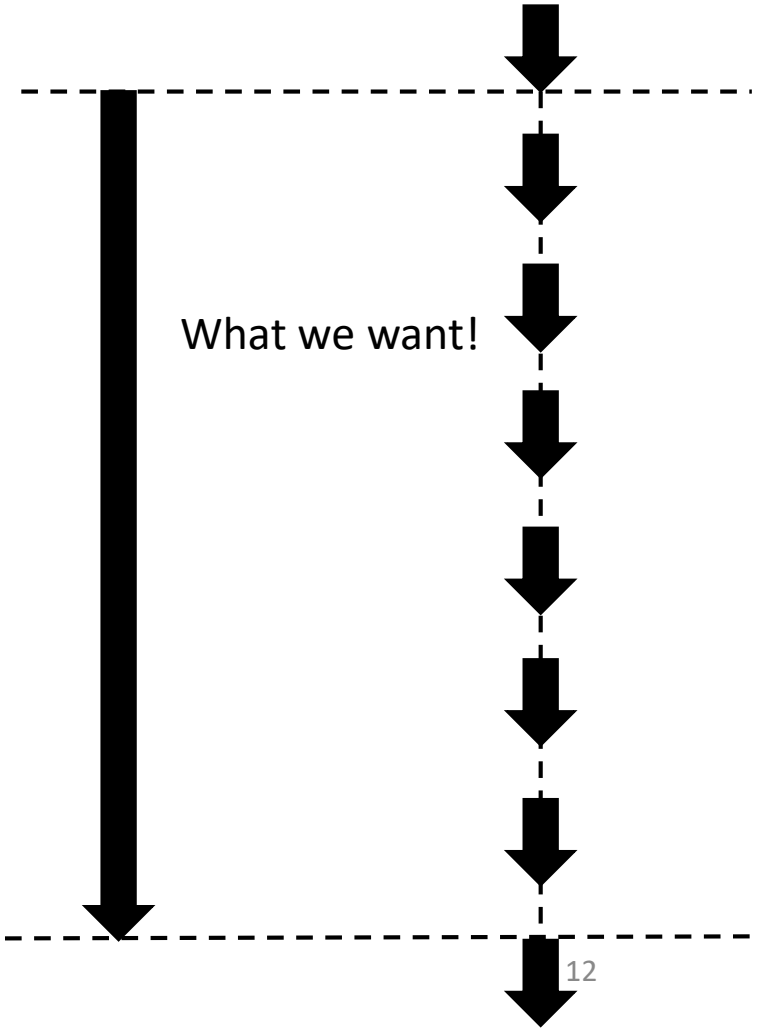
Running http://webglsamples.org/electricflower/electricflower.html
with sssp OpenCL application in background on Intel HD5500

Multi-tasking

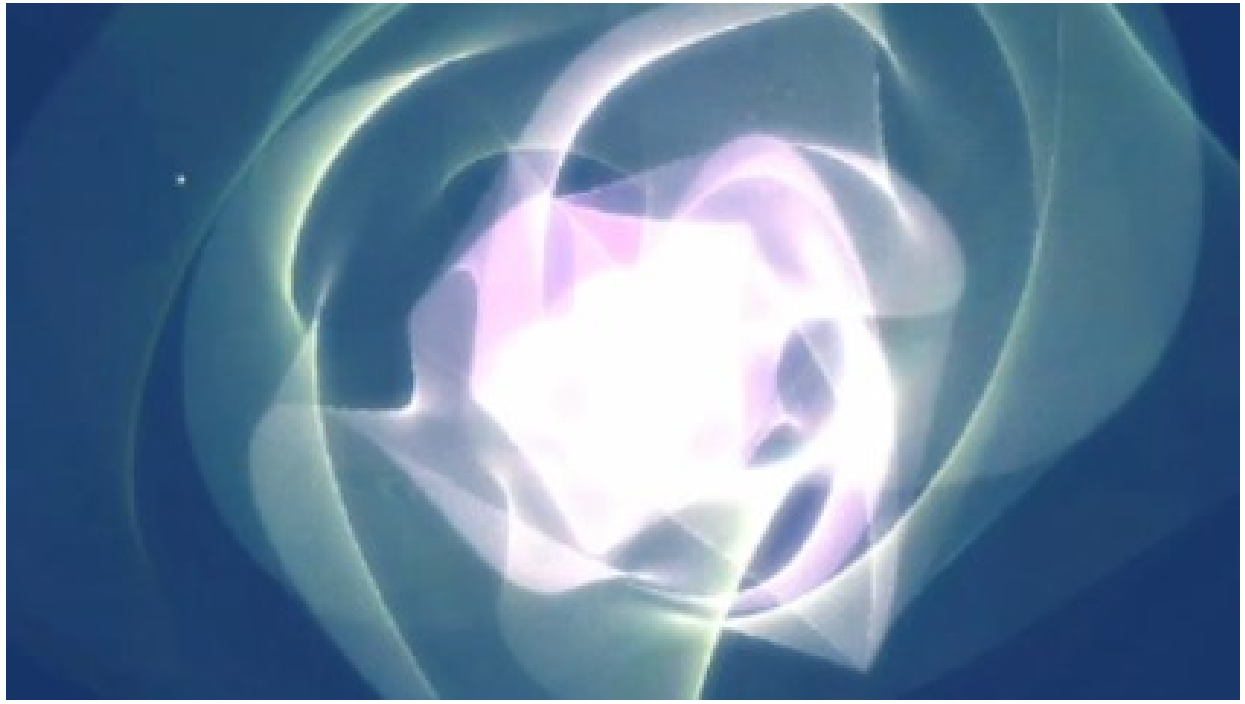


Running http://webglsamples.org/electricflower/electricflower.html
with sssp OpenCL application in background on Intel HD5500

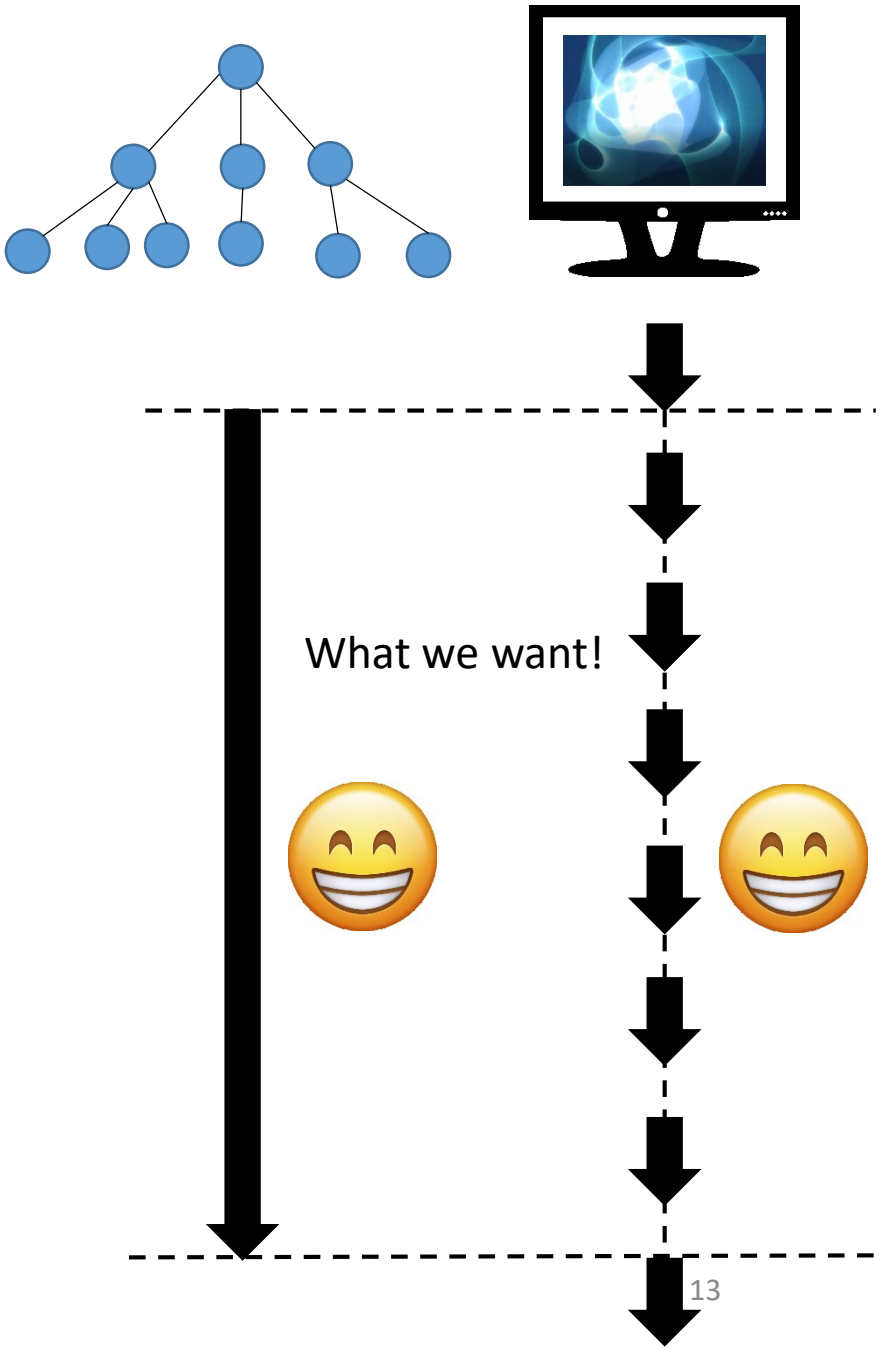




Multi-tasking



Running http://webglsamples.org/electricflower/electricflower.html
with sssp OpenCL application in background on Intel HD5500



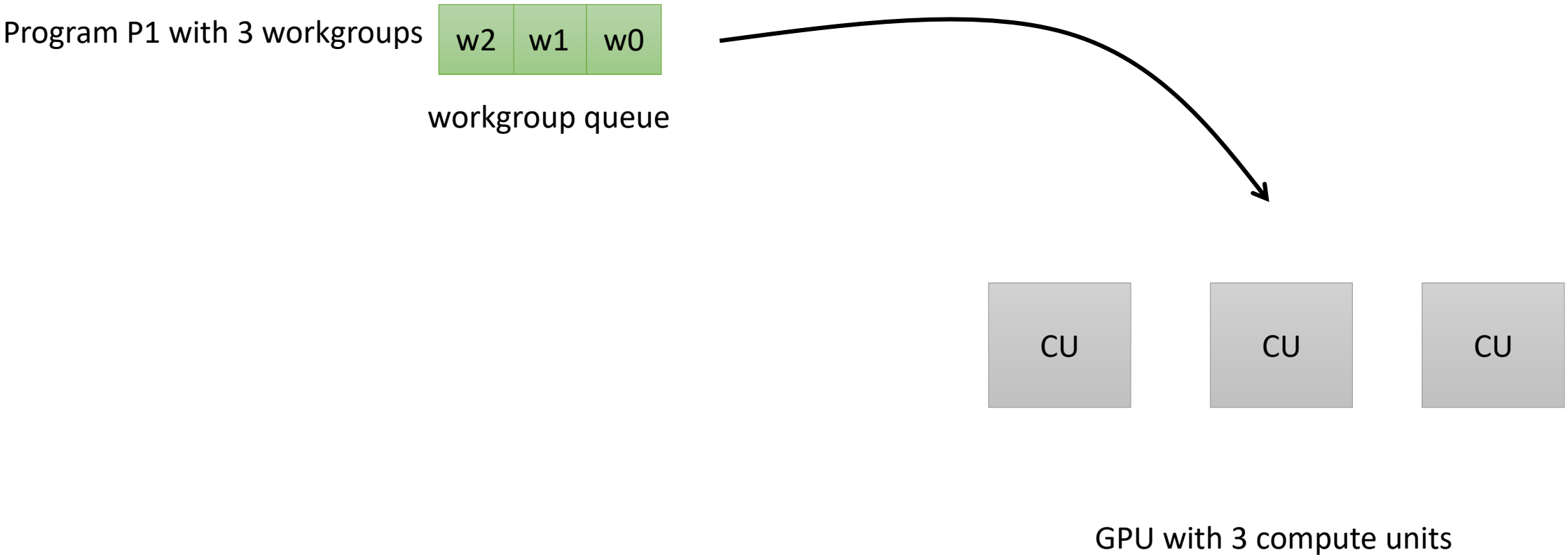
Multi-tasking

Program P1 with 3 workgroups

workgroup queue

GPU with 3 compute units

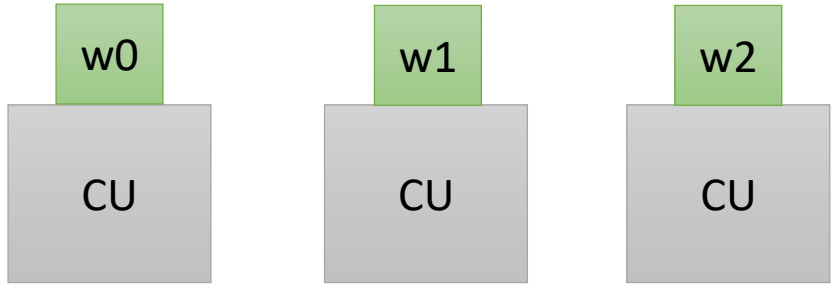
Multi-tasking



Multi-tasking

Program P1 with 3 workgroups

workgroup queue



GPU with 3 compute units

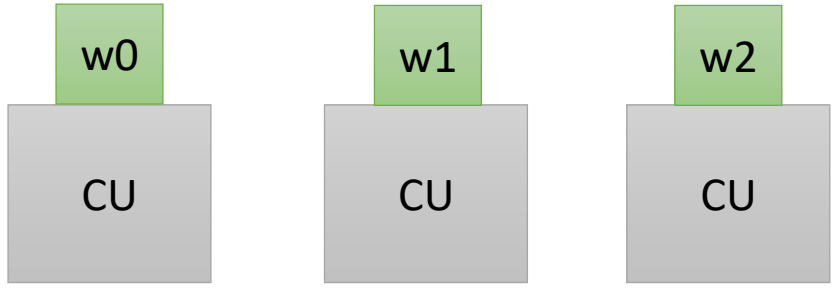
Multi-tasking

Program P1 with 3 workgroups

workgroup queue

Program P2 with 3 workgroups

workgroup queue



GPU with 3 compute units

Multi-tasking

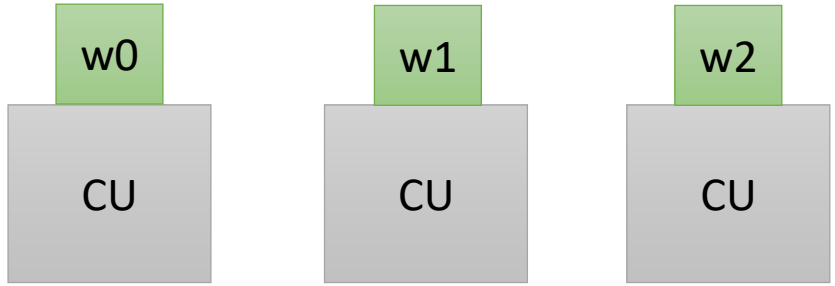
Program P1 with 3 workgroups

workgroup queue

Program P2 with 3 workgroups

workgroup queue

Current GPUs just have P2 wait for P1, causing frame skips



GPU with 3 compute units

Multi-tasking

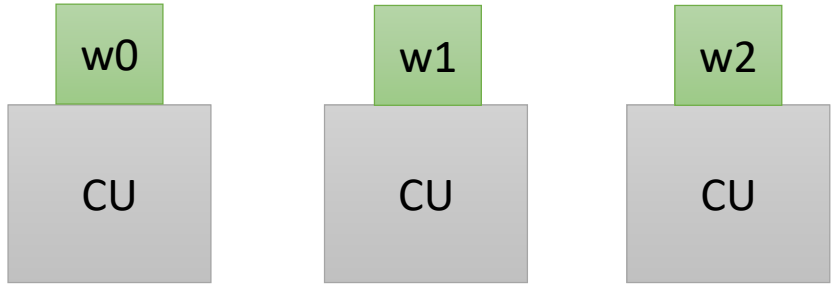
Program P1 with 3 workgroups

workgroup queue

Program P2 with 3 workgroups

workgroup queue

What about preemption?



GPU with 3 compute units

Multi-tasking

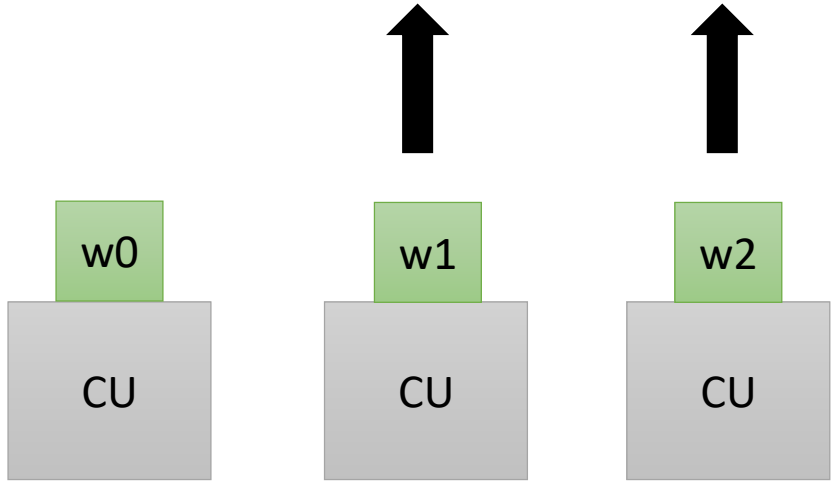
Program P1 with 3 workgroups

workgroup queue

Program P2 with 3 workgroups

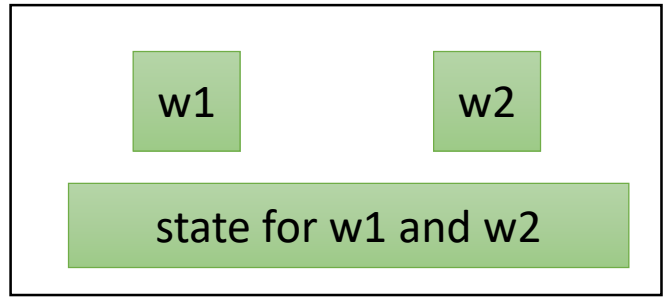
workgroup queue

What about preemption?



GPU with 3 compute units

Multi-tasking



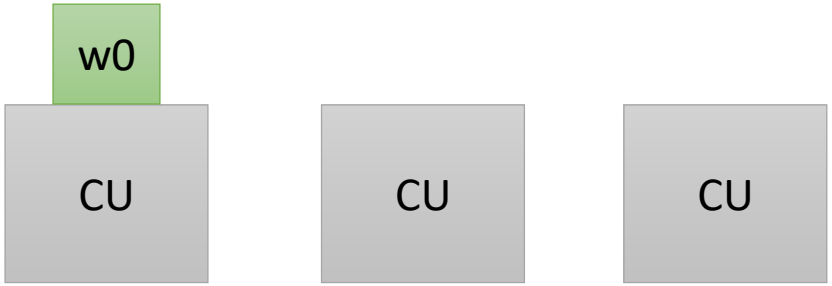
Program P1 with 3 workgroups

workgroup queue

Program P2 with 3 workgroups

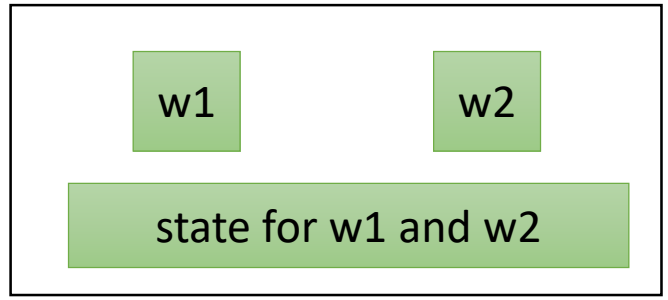
workgroup queue

What about preemption?



GPU with 3 compute units

Multi-tasking



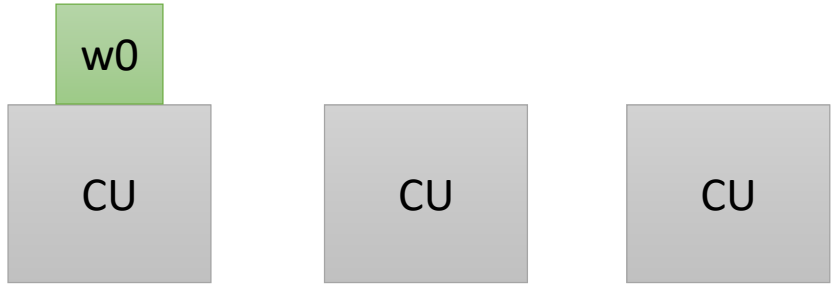
Program P1 with 3 workgroups

workgroup queue

P2 can execute on these 2 CUs

Program P2 with 3 workgroups

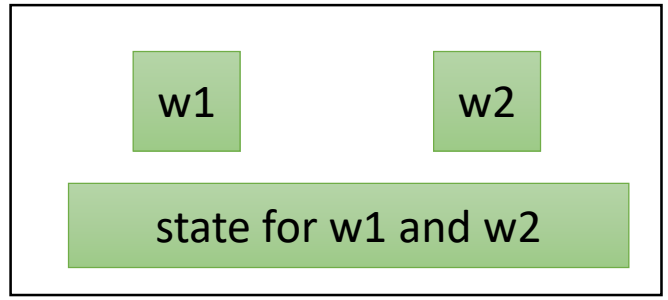
workgroup queue



What about preemption?

GPU with 3 compute units

Multi-tasking



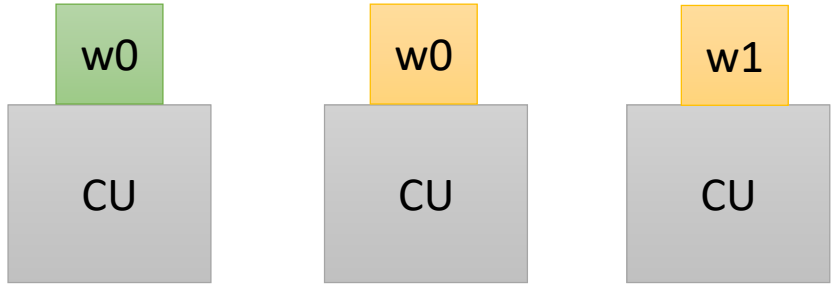
Program P1 with 3 workgroups

workgroup queue

P2 can execute on these 2 CUs

Program P2 with 3 workgroups

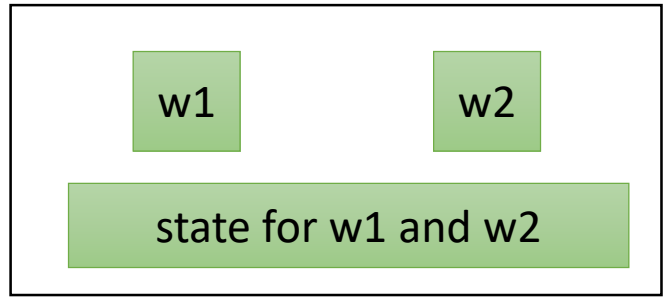
workgroup queue



What about preemption?

GPU with 3 compute units

Multi-tasking



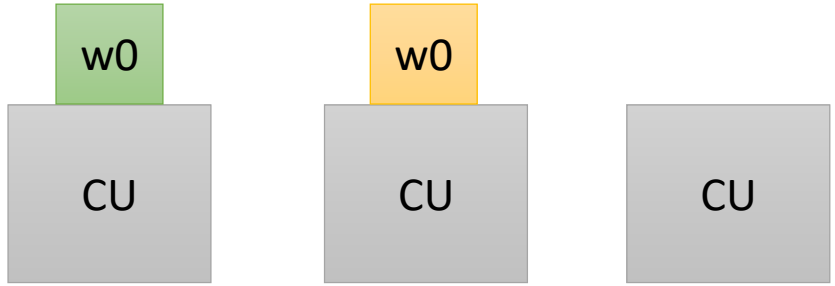
Program P1 with 3 workgroups

workgroup queue

P2 can execute on these 2 CUs

Program P2 with 3 workgroups

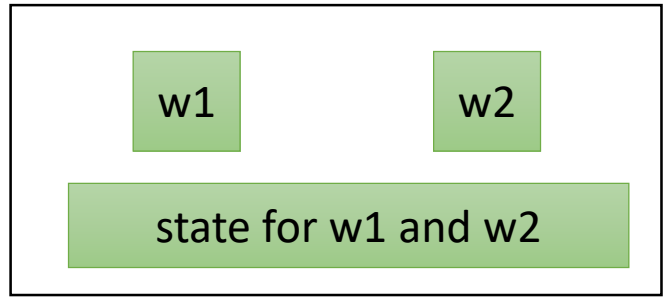
workgroup queue



What about preemption?

GPU with 3 compute units

Multi-tasking



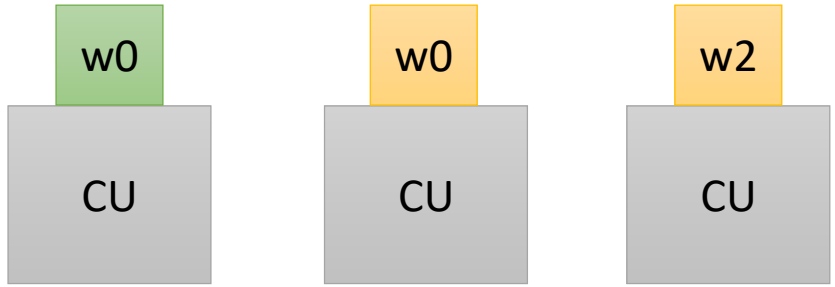
Program P1 with 3 workgroups

workgroup queue

P2 can execute on these 2 CUs

Program P2 with 3 workgroups

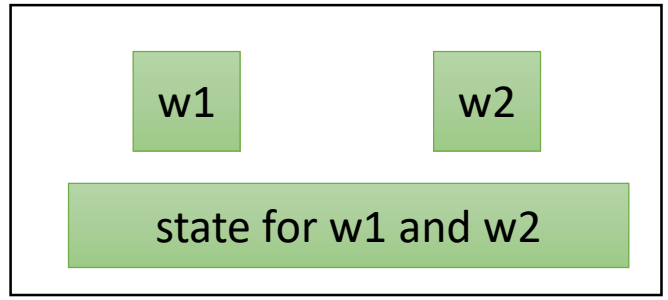
workgroup queue



What about preemption?

GPU with 3 compute units

Multi-tasking



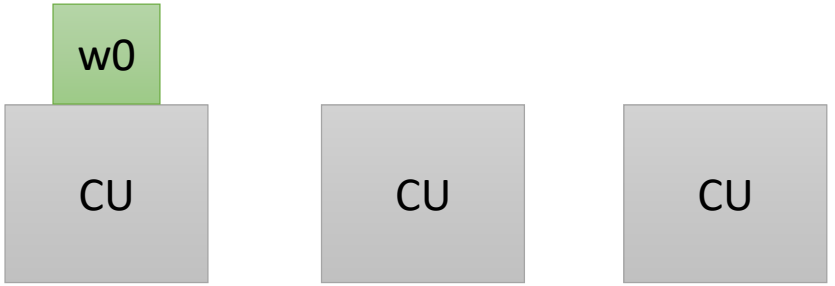
Program P1 with 3 workgroups

workgroup queue

Program P2 with 3 workgroups

workgroup queue

What about preemption?



GPU with 3 compute units

Multi-tasking

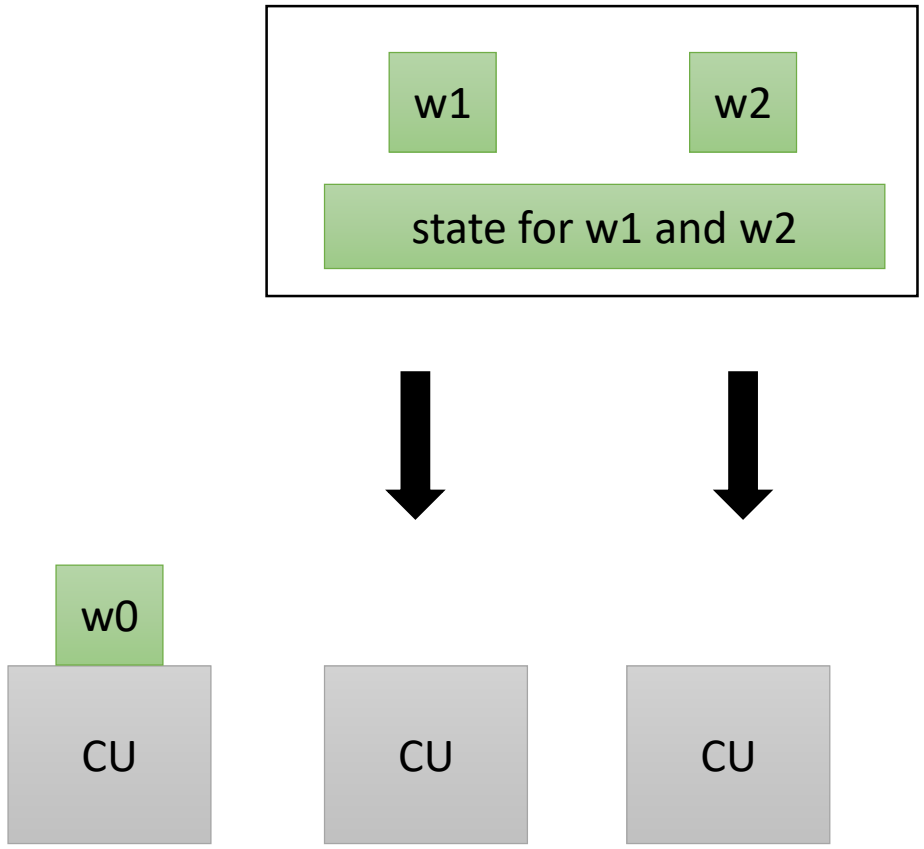
Program P1 with 3 workgroups

workgroup queue

Program P2 with 3 workgroups

workgroup queue

What about preemption?



GPU with 3 compute units

Multi-tasking

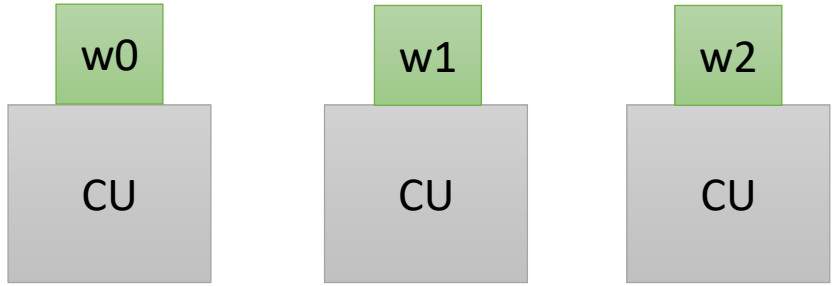
Program P1 with 3 workgroups

workgroup queue

Program P2 with 3 workgroups

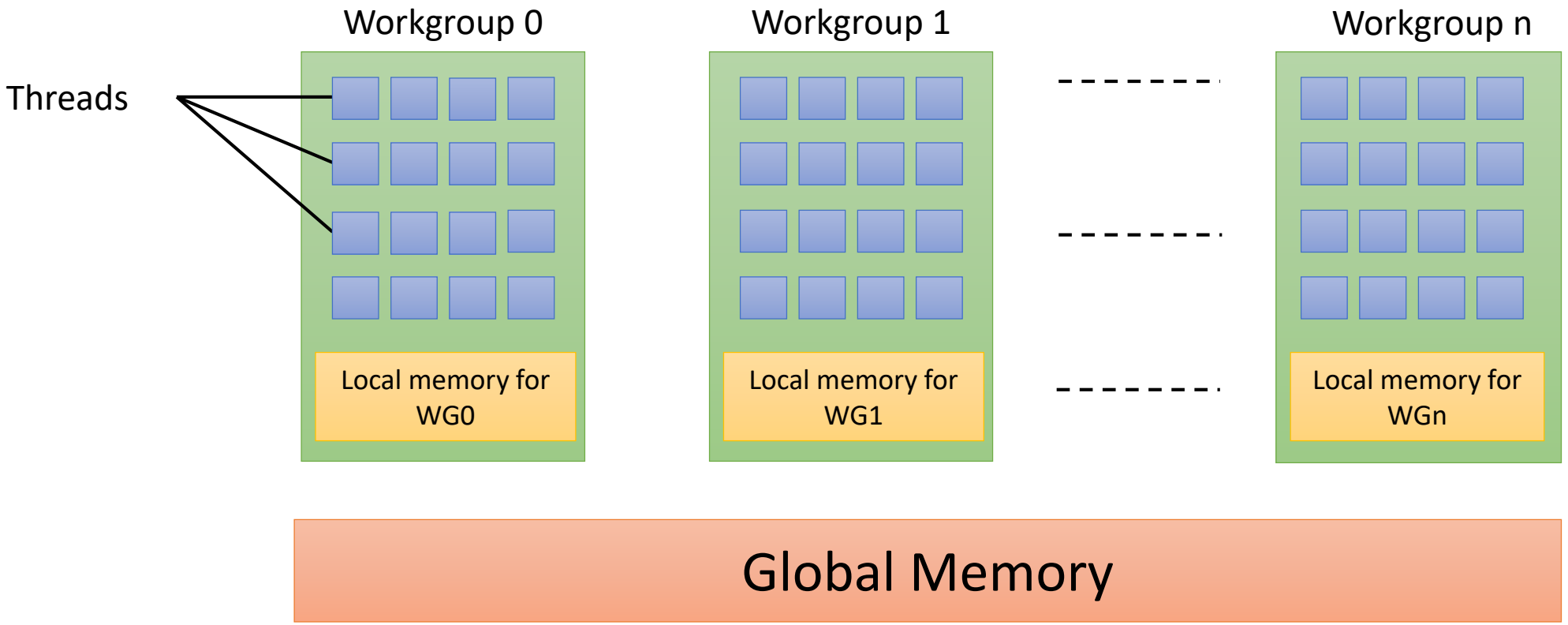
workgroup queue

Easy, just need to save state!



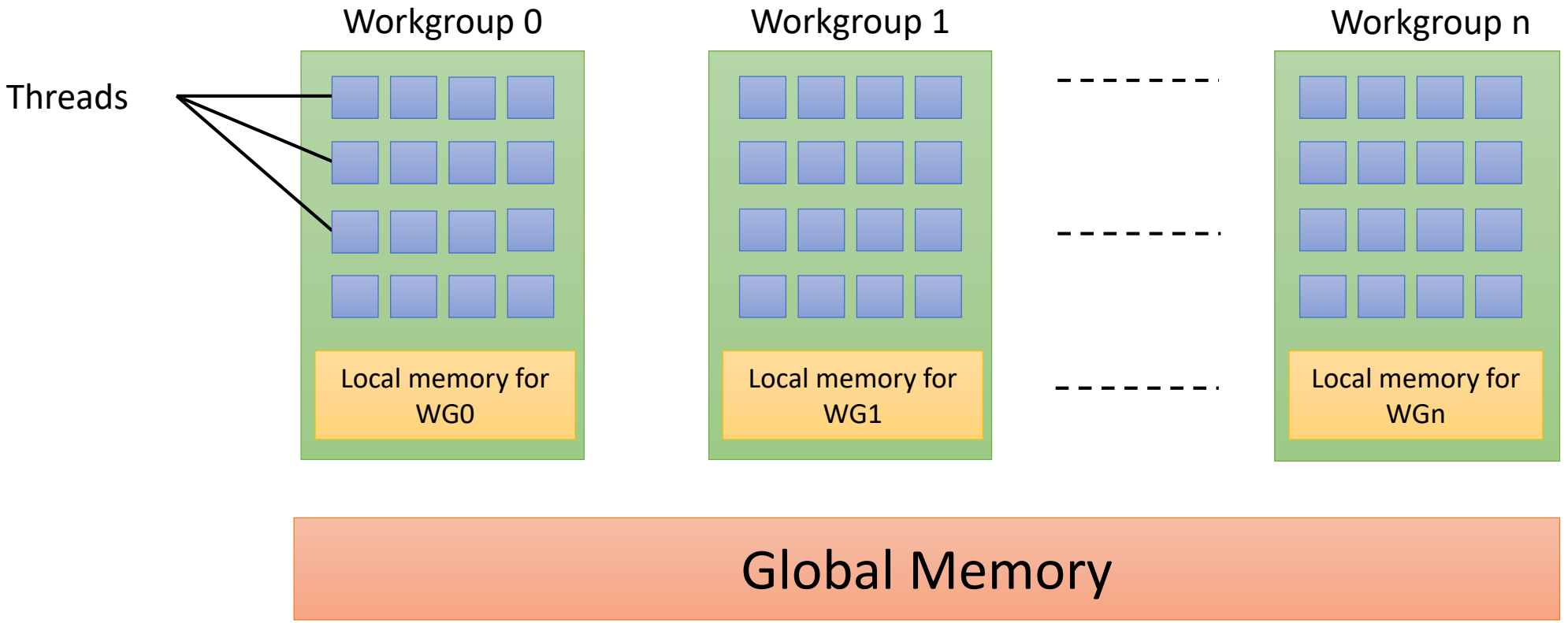
GPU with 3 compute units

GPU programming model



GPU programming model

Saving state for a workgroup is **EXPENSIVE**! includes PC, registers for all threads and local memory

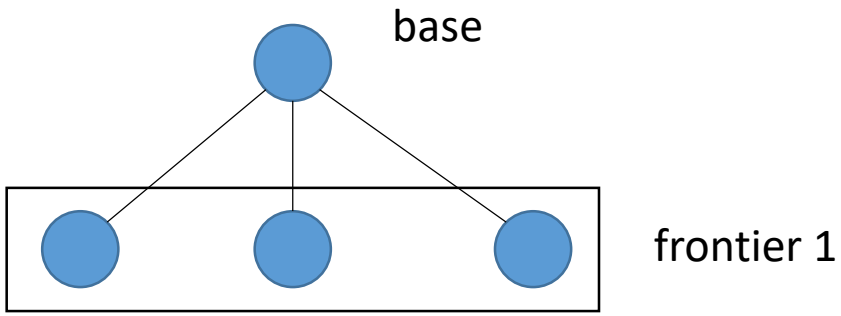


Example application

- Example: Frontier based sssp graph traversal

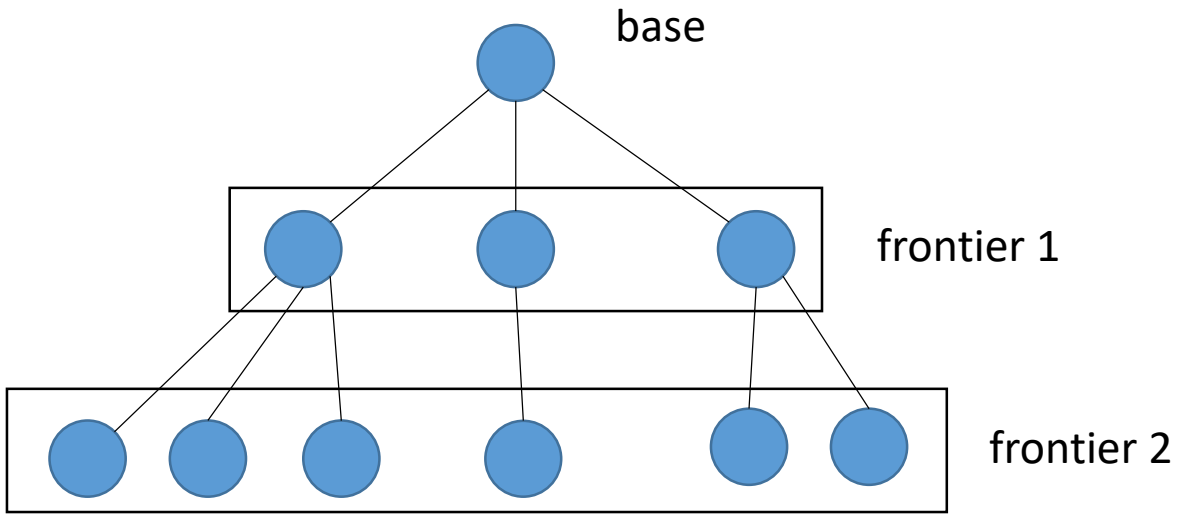
Example application

- Example: Frontier based sssp graph traversal



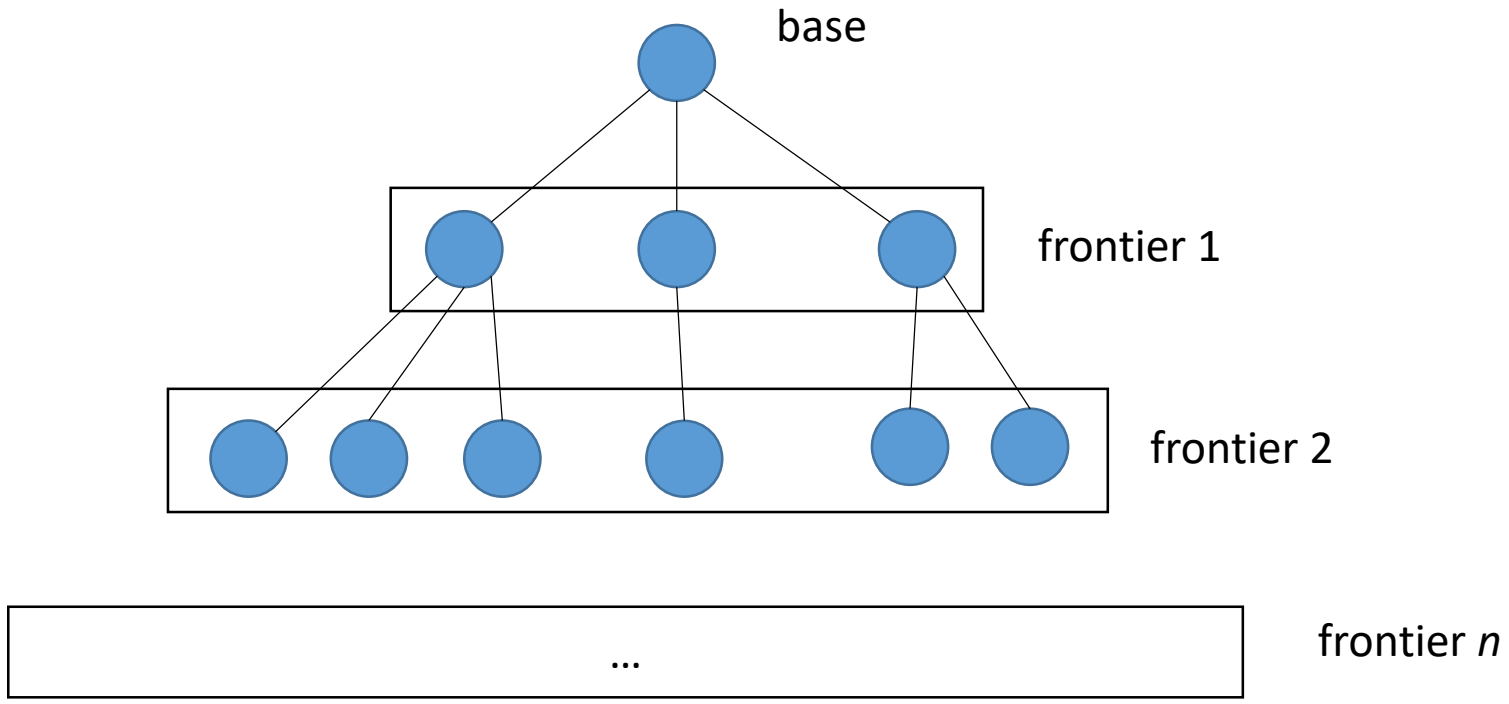
Example application

- Example: Frontier based sssp graph traversal



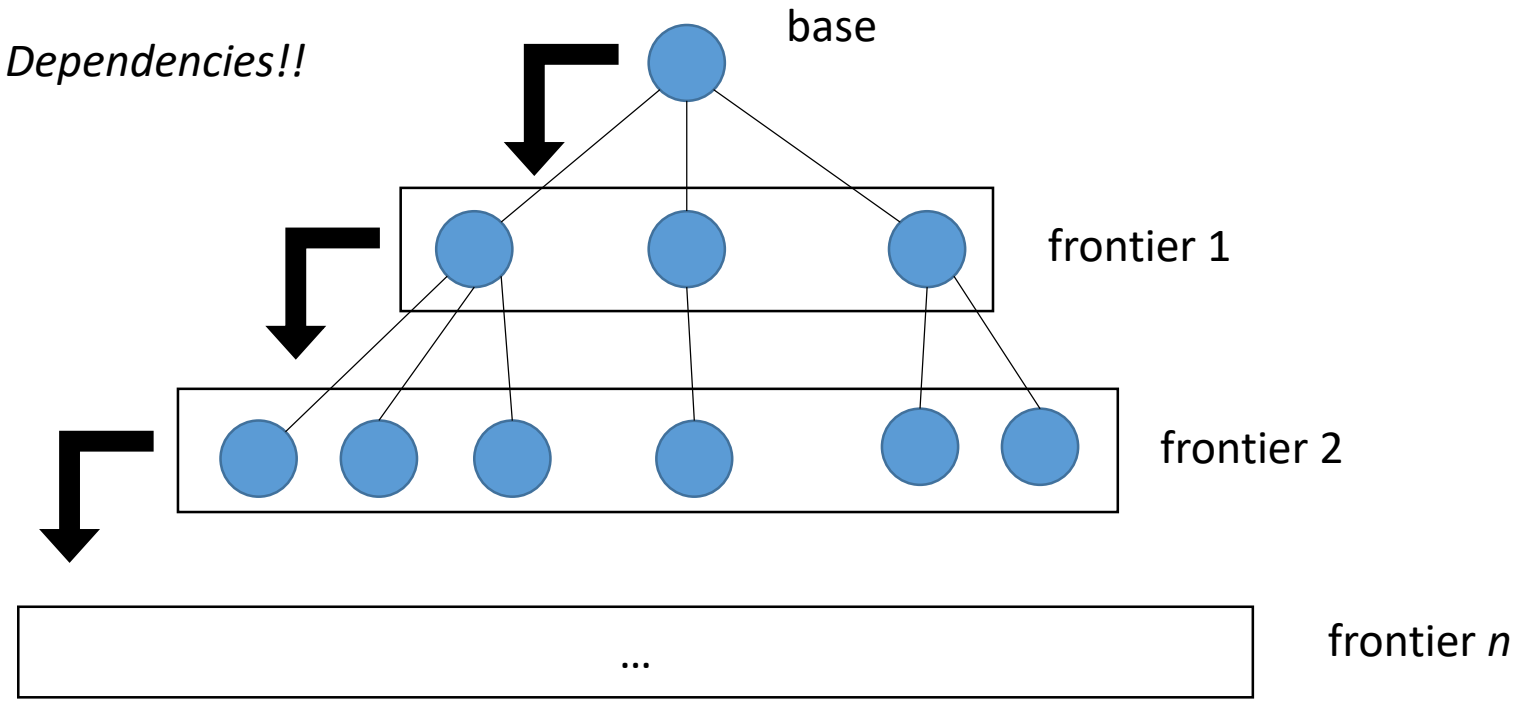
Example application

- Example: Frontier based sssp graph traversal



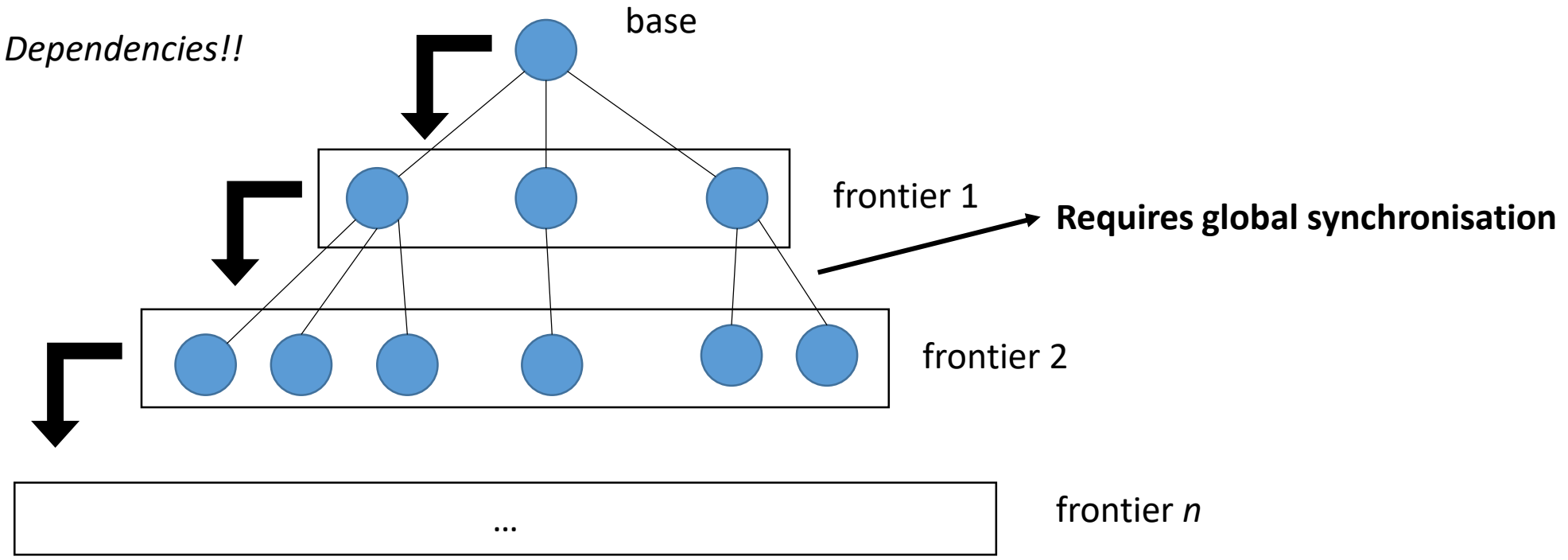
Example application

- Example: Frontier based sssp graph traversal



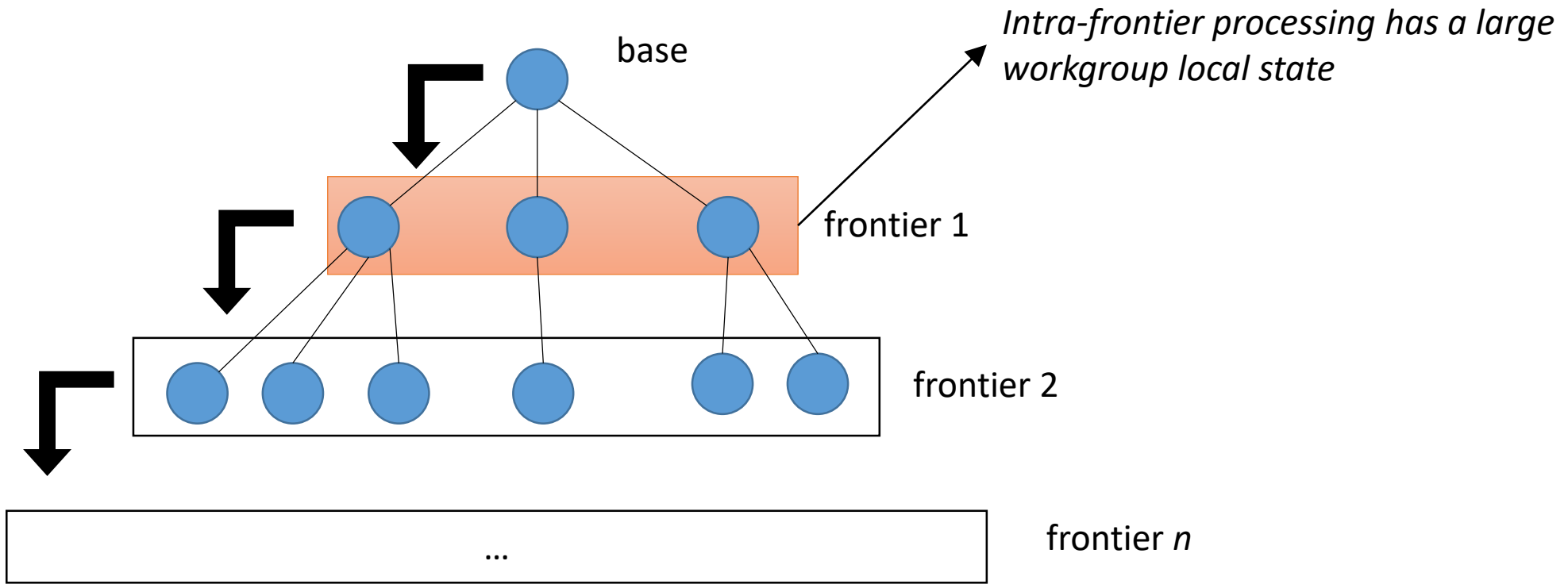
Example application

- Example: Frontier based sssp graph traversal



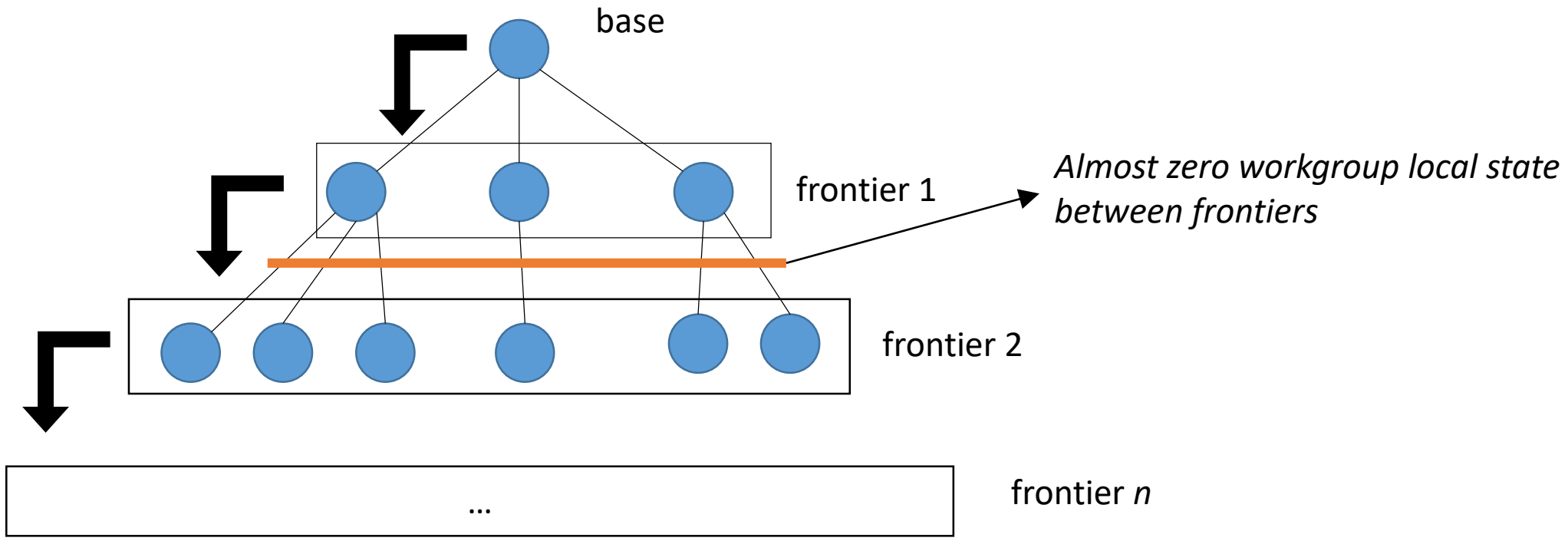
Example application

- Example: Frontier based sssp graph traversal



Example application

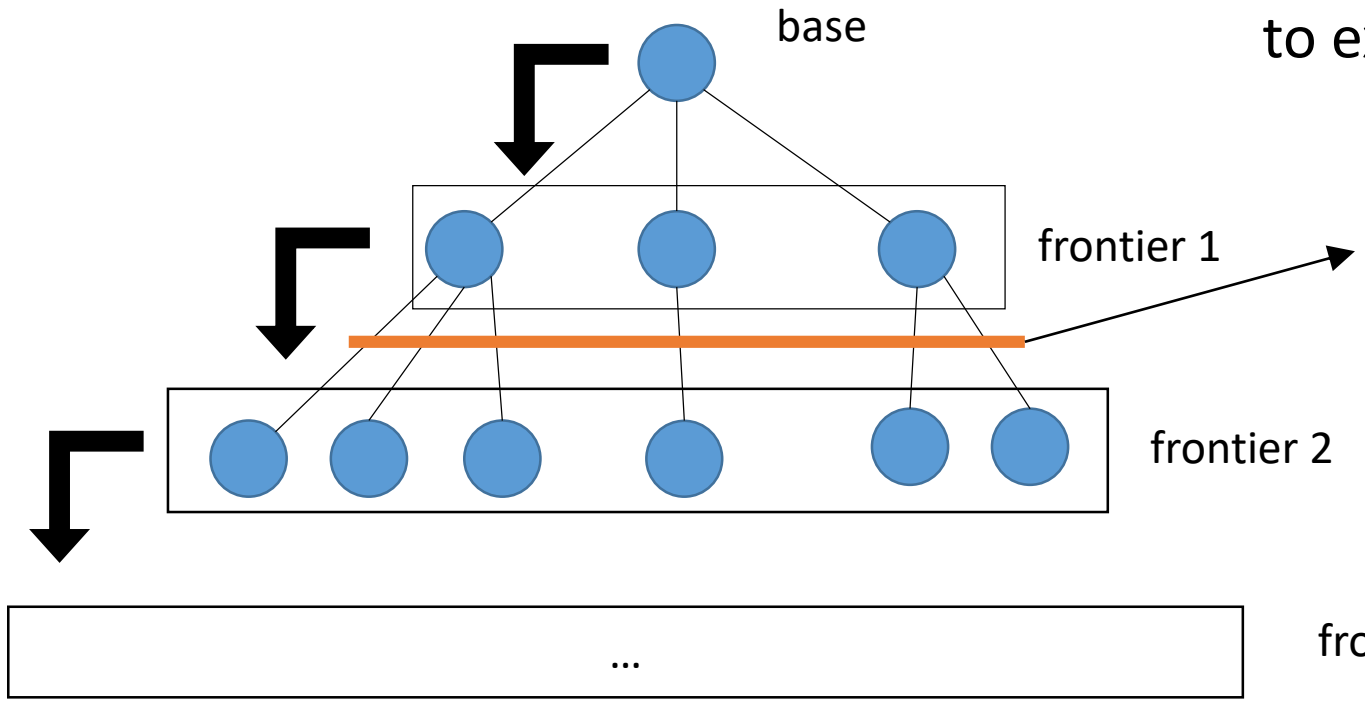
- Example: Frontier based sssp graph traversal



Example application

- Example: Frontier based sssp graph traversal

What if programmers had a way to express this?



Almost zero workgroup local state between frontiers

Our solution

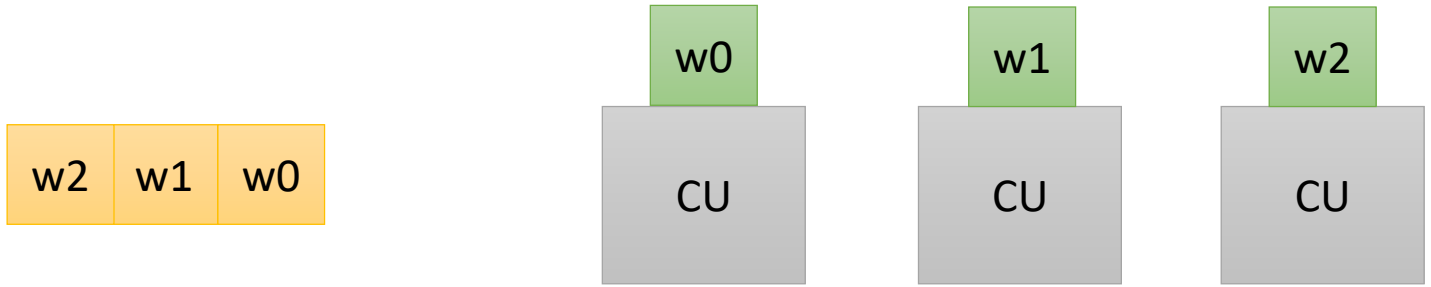
- **Cooperative kernels**: Long running kernels (e.g. graph traversal) that could interrupt short running kernels (e.g. graphics)

- Cooperative kernels must *share* resources via new programming constructs which enable

- Framework provides cooperative kernels *guaranteed* execution

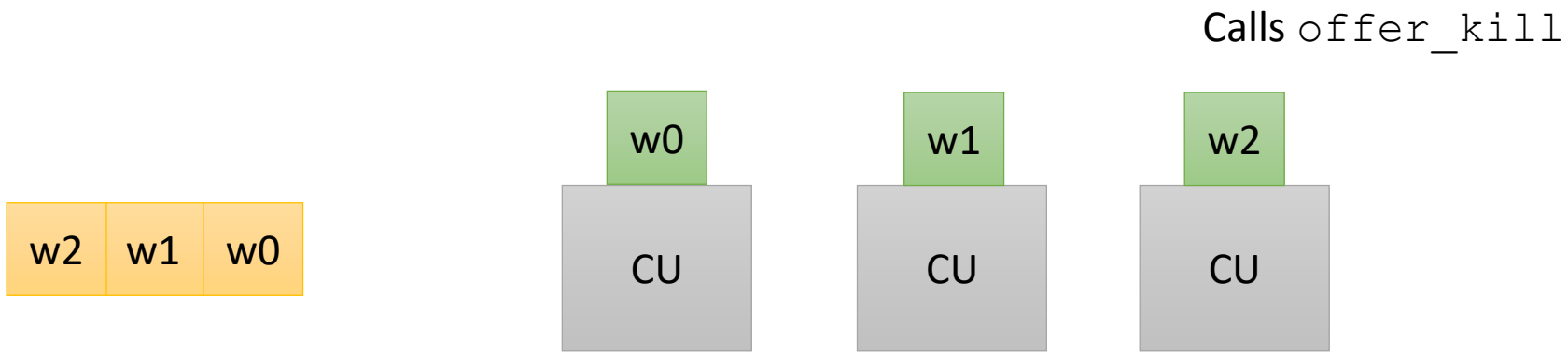
Cooperative kernel workgroup functions

- `offer_kill`
  - Scheduler can kill calling workgroup if needed
  - Kills workgroups in descending id order



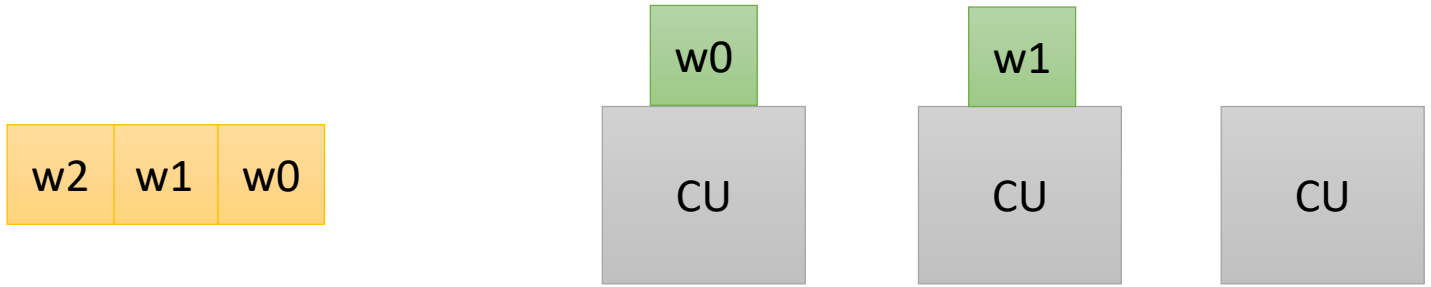
Cooperative kernel workgroup functions

- `offer_kill`
  - Scheduler can kill calling workgroup if needed
  - Kills workgroups in descending id order



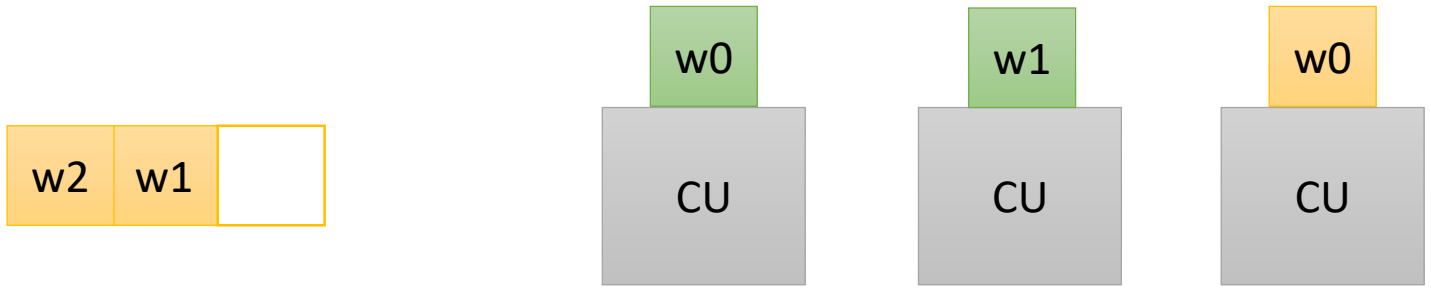
Cooperative kernel workgroup functions

- `offer_kill`
  - Scheduler can kill calling workgroup if needed
  - Kills workgroups in descending id order



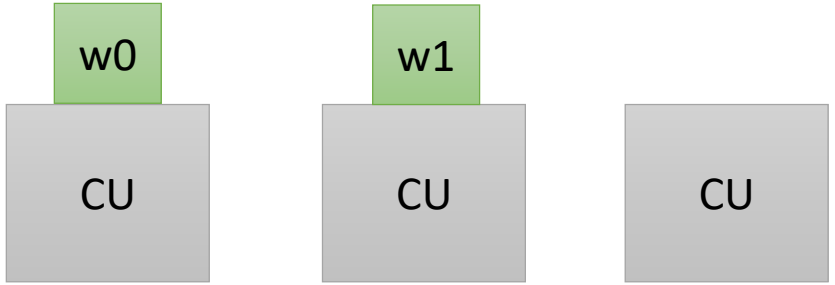
Cooperative kernel workgroup functions

- `offer_kill`
  - Scheduler can kill calling workgroup if needed
  - Kills workgroups in descending id order



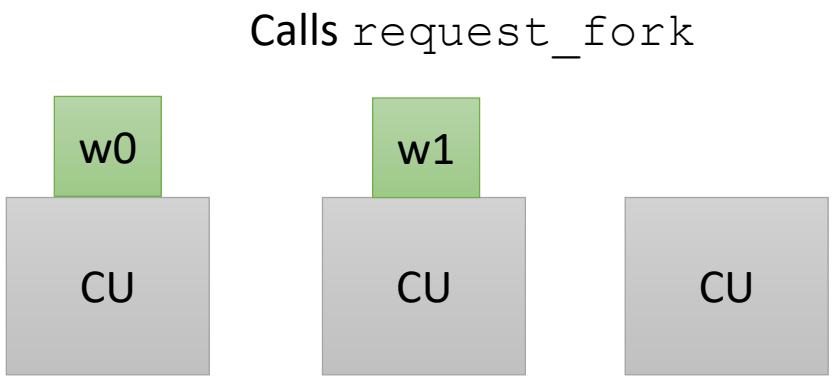
Cooperative kernel workgroup functions

- `request_fork`
  - If there are available resources, fork workgroups
  - Assigns new ids to forked workgroups



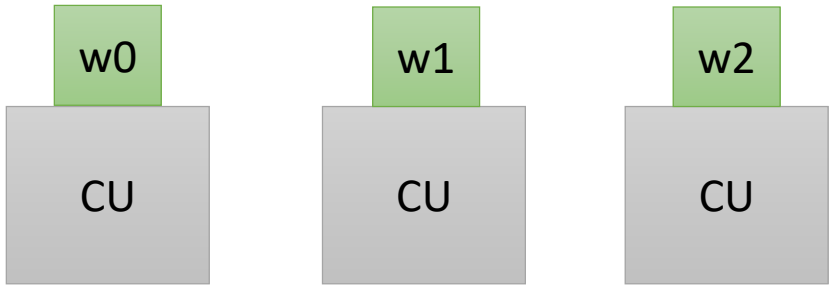
Cooperative kernel workgroup functions

- `request_fork`
  - If there are available resources, fork workgroups
  - Assigns new ids to forked workgroups



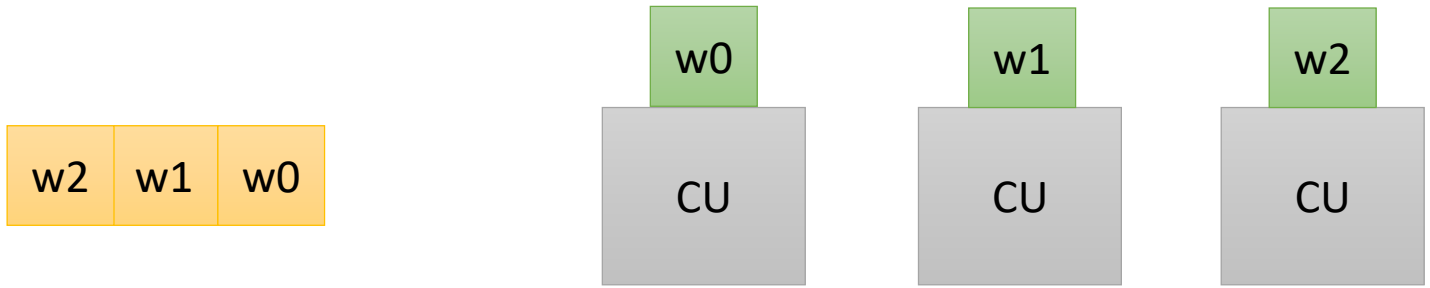
Cooperative kernel workgroup functions

- `request_fork`
  - If there are available resources, fork workgroups
  - Assigns new ids to forked workgroups



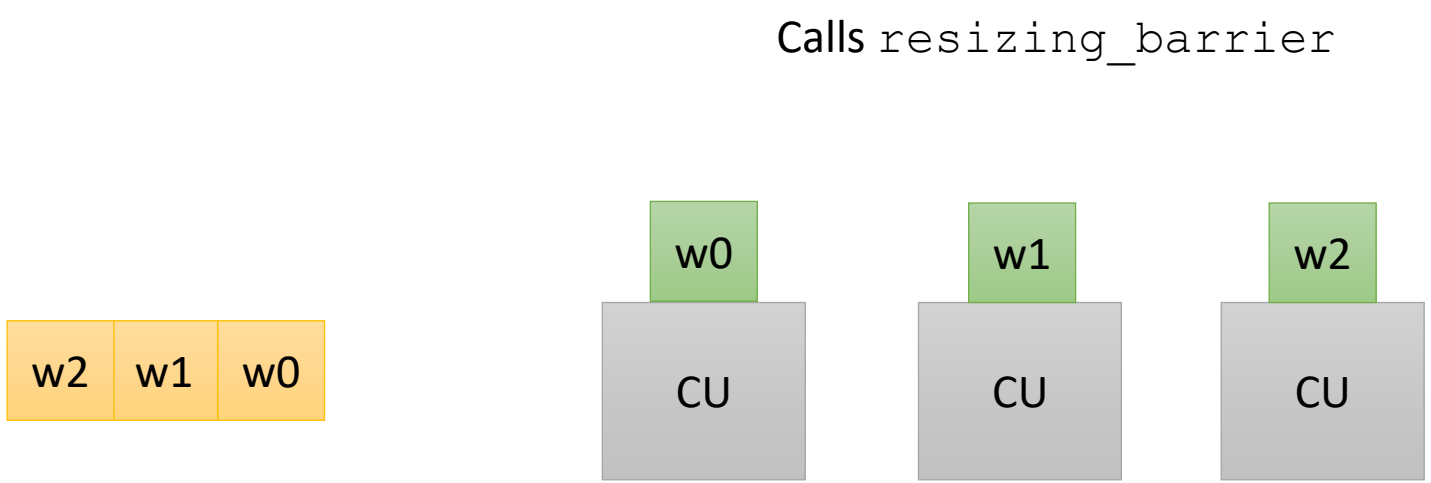
Cooperative kernel workgroup functions

- `resizing_barrier`
  - A barrier where workgroups may leave/join



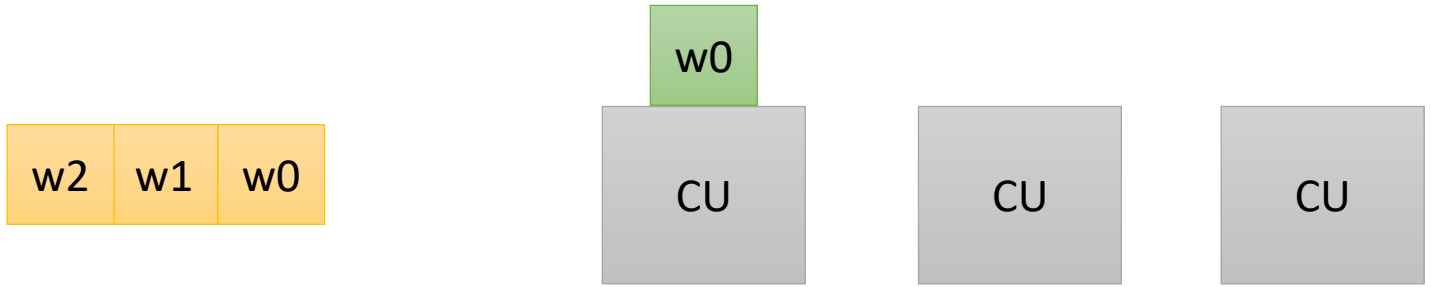
Cooperative kernel workgroup functions

- `resizing_barrier`
  - A barrier where workgroups may leave/join



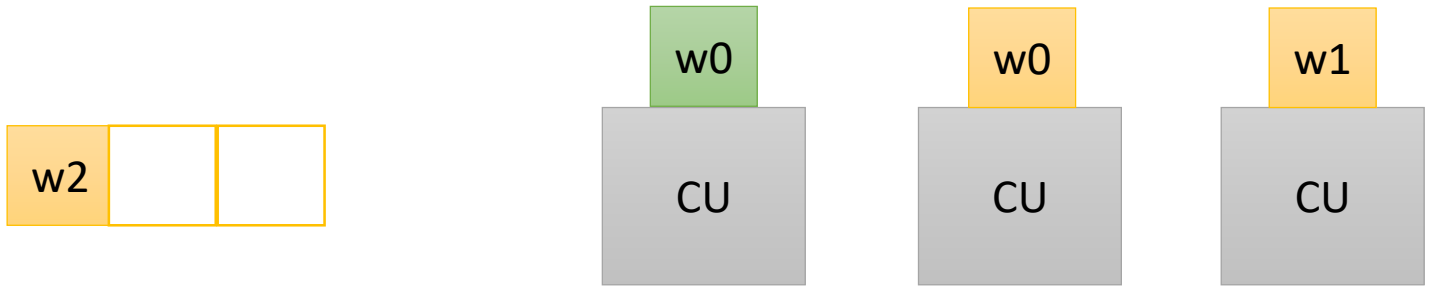
Cooperative kernel workgroup functions

- `resizing_barrier`
  - A barrier where workgroups may leave/join



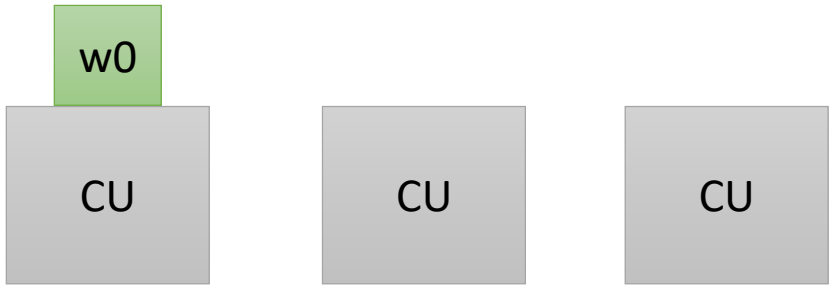
Cooperative kernel workgroup functions

- `resizing_barrier`
  - A barrier where workgroups may leave/join



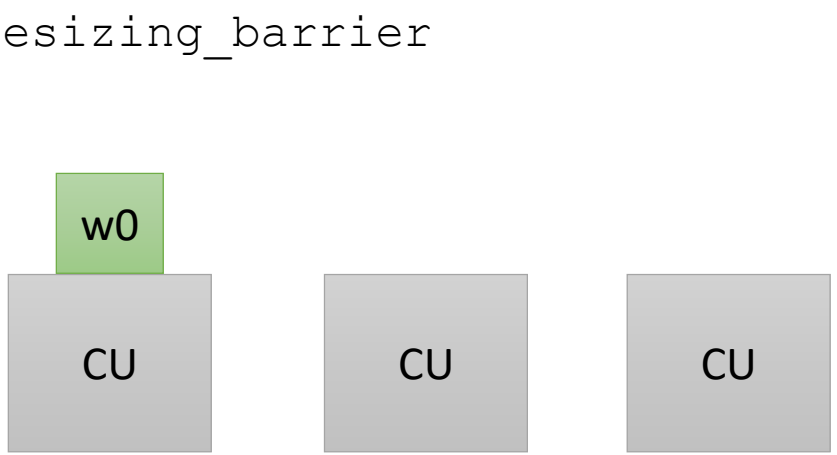
Cooperative kernel workgroup functions

- `resizing_barrier`
  - A barrier where workgroups may leave/join



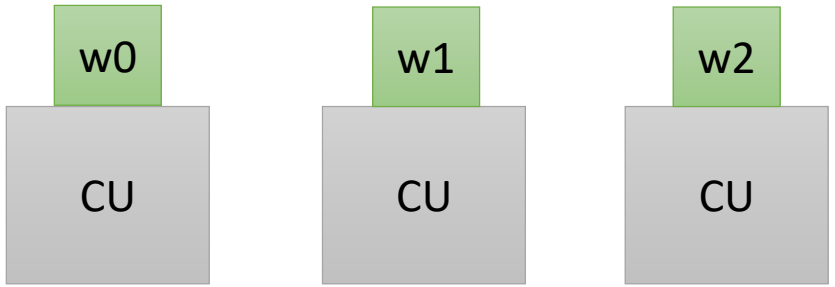
Cooperative kernel workgroup functions

- `resizing_barrier`
  - A barrier where workgroups may leave/join

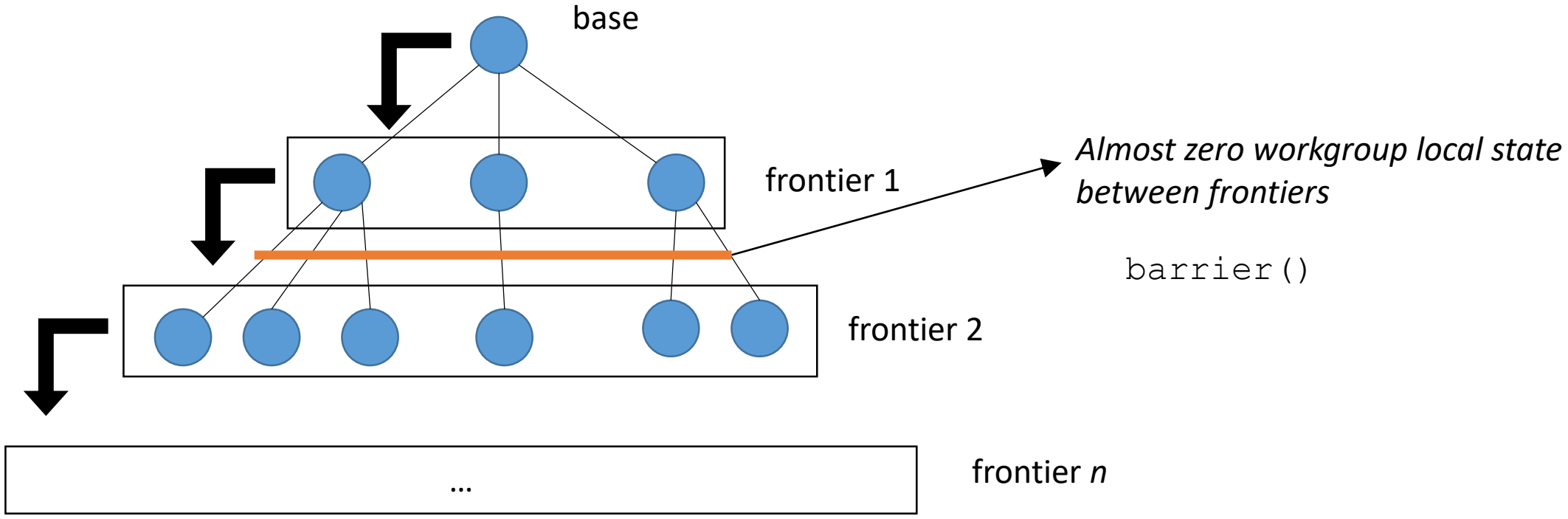


Cooperative kernel workgroup functions

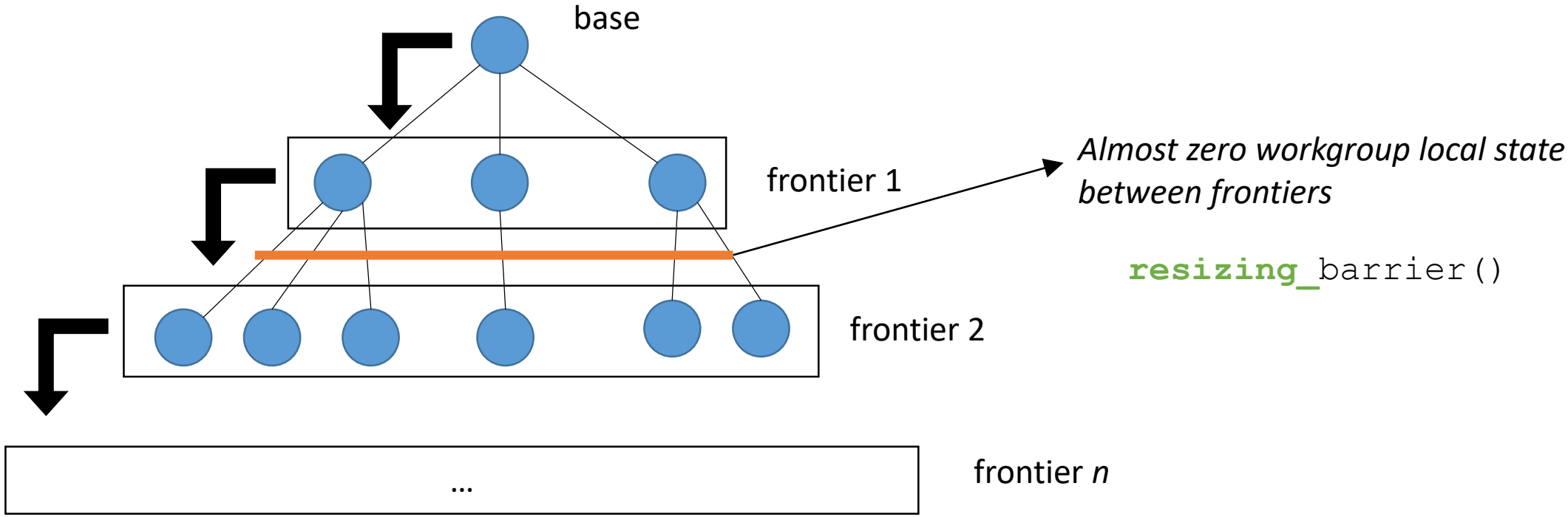
- `resizing_barrier`
  - A barrier where workgroups may leave/join



Example application



Example application

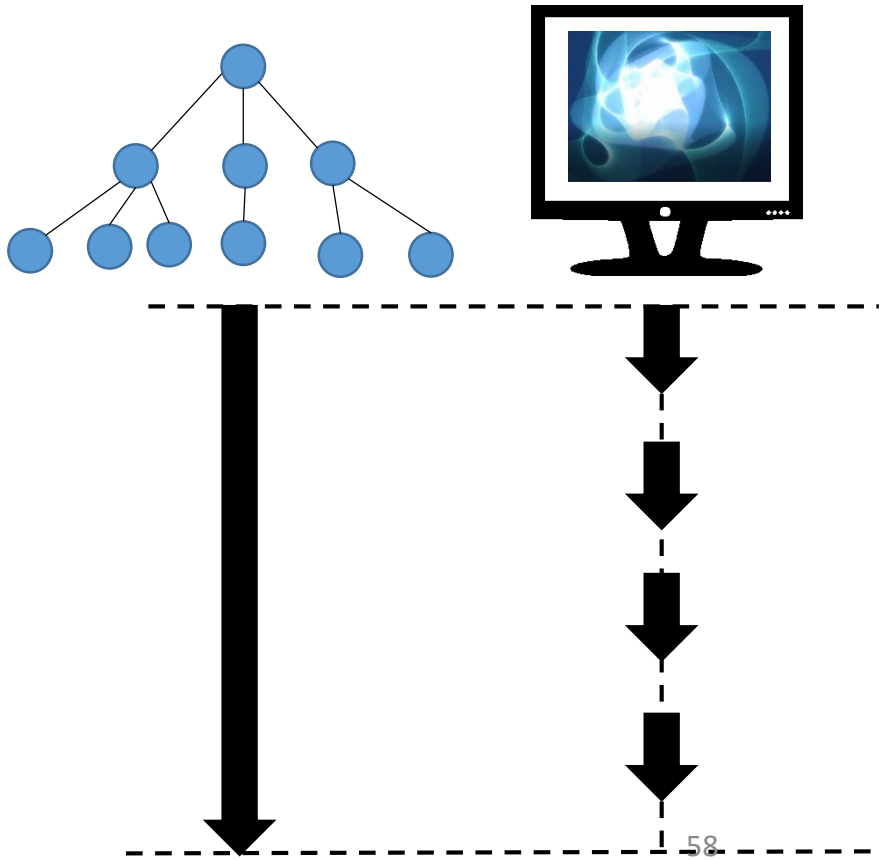


Evaluation (ease of programming?)

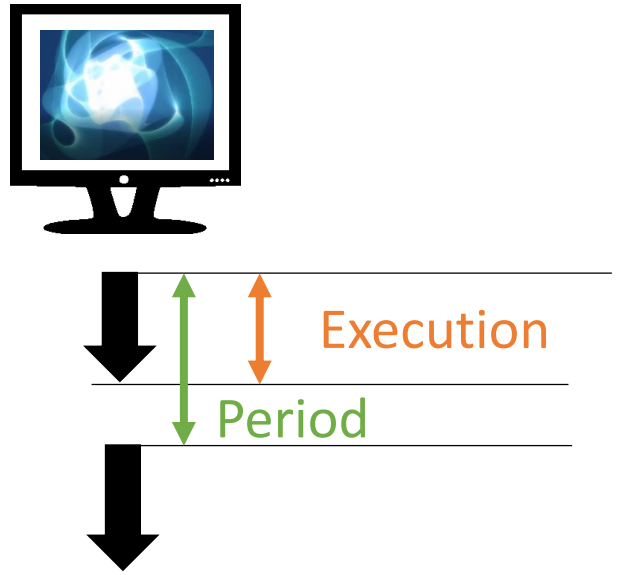
- Porting existing applications to use cooperative kernels
- Two types of applications:
  - 6 global barrier applications
    - Changed all global barriers to resizing barriers
  - 2 work stealing applications
    - added 1 call to offer_kill and request_fork
- DSL can automatically generates cooperative kernels

Evaluation (does it actually work?)

- Prototype implementation models two task systems
- We experiment multi-tasking:
  - 8 long-running applications
  - 3 graphics workloads
- We run on Intel Iris 6100 GPU
- Prototype overhead: 10% (upper-bound!)



Graphics workloads

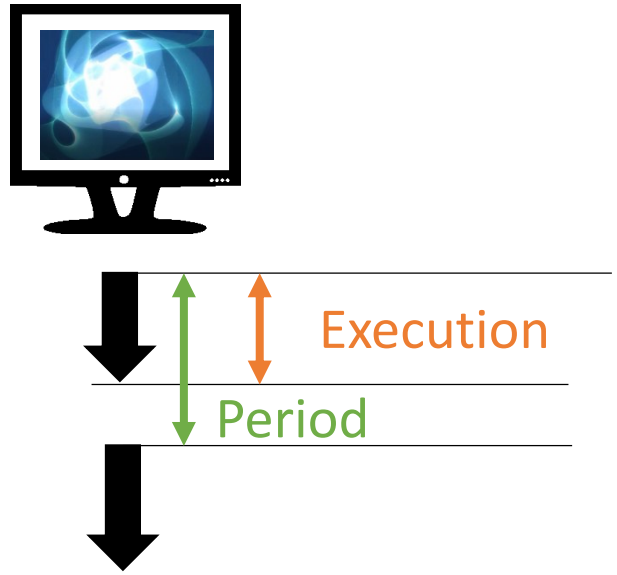


Can we reach the deadlines for smooth graphics?

Workload	Period	Execution (full GPU)	Cooperative overhead	Compute units
Light	70 ms	3 ms		
Medium	40 ms	3 ms		
Heavy	40 ms	10 ms		

Geo. mean

Graphics workloads

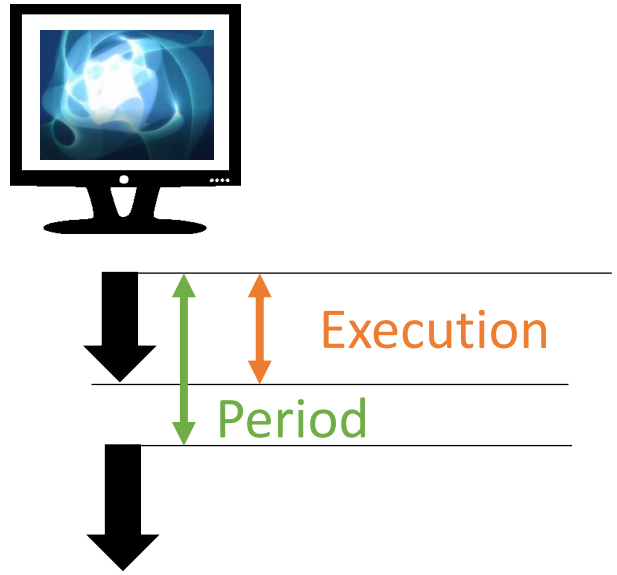


Can we reach the deadlines for smooth graphics?

Workload	Period	Execution (full GPU)	Cooperative overhead	Compute units
Light	70 ms	3 ms	1.00x	25%
Medium	40 ms	3 ms		
Heavy	40 ms	10 ms		

Geo. mean

Graphics workloads

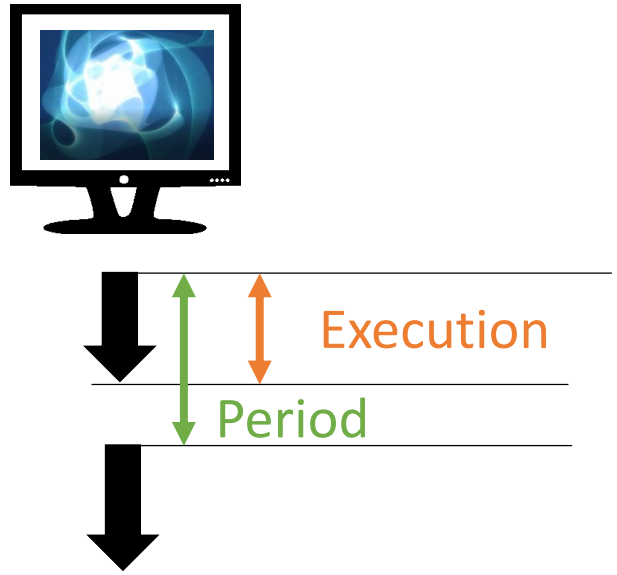


Can we reach the deadlines for smooth graphics?

Workload	Period	Execution (full GPU)	Cooperative overhead	Compute units
Light	70 ms	3 ms	1.00x	25%
Medium	40 ms	3 ms	1.03x	25%
Heavy	40 ms	10 ms		

Geo. mean

Graphics workloads



Can we reach the deadlines for smooth graphics?

Workload	Period	Execution (full GPU)	Cooperative overhead	Compute units
Light	70 ms	3 ms	1.00x	25%
Medium	40 ms	3 ms	1.03x	25%
Heavy	40 ms	10 ms	1.28x	50%
			Geo. mean	

Cooperative Kernels: GPU Multitasking for Blocking Algorithms

- GPU multitasking is a complex unsolved issue
- Our solution provides **3 new primitives** for programmers to interact with scheduler for **efficient GPU multitasking**
- Prototype implementation achieves soft real-time constraints for compute + graphics

Tyler Sorensen https://www.doc.ic.ac.uk/~trs15/