JOURNEY TO THE CENTRE OF THE OPENCL MEMORY MODEL

John Wickerson
Imperial College London

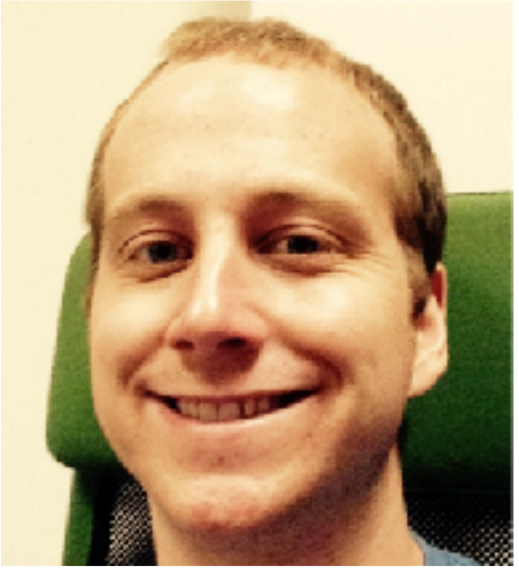
COLLABORATORS

Mark Batty

George Constantinides

Nadesh Ramanathan

Alastair Donaldson



Tyler Sorensen

Brad Beckmann

THIS TALK

THIS TALK

- Weak memory

THIS TALK

- Weak memory
- Formalising the OpenCL memory model

THIS TALK

- Weak memory
- Formalising the OpenCL memory model
- Implementing the OpenCL memory model on GPUs

THIS TALK

- Weak memory
- Formalising the OpenCL memory model
- Implementing the OpenCL memory model on GPUs
- Implementing the OpenCL memory model on FPGAs

WEAK MEMORY

WEAK MEMORY

```
MOV [x] 1   || MOV [y] 1

MOV r0 [y]  || MOV r1 [x]
```

WEAK MEMORY

```
MOV [x] 1    || MOV [y] 1

MOV r0 [y]   || MOV r1 [x]
```

r0=1

r1=1

WEAK MEMORY

```
MOV [x] 1     ||  MOV [y] 1

MOV r0 [y]    ||  MOV r1 [x]
```

```
r0=1      r0=0
r1=1      r1=1
```

WEAK MEMORY

```
MOV [x] 1    ║ MOV [y] 1

MOV r0 [y]   ║ MOV r1 [x]
```

```
r0=1        r0=0        r0=1
r1=1        r1=1        r1=0
```

WEAK MEMORY

```
MOV [x] 1    ||  MOV [y] 1
MOV r0 [y]   ||  MOV r1 [x]
```

$$\underbrace{\begin{matrix} \texttt{r0=1} & \quad & \texttt{r0=0} & \quad & \texttt{r0=1} \\ \texttt{r1=1} & \quad & \texttt{r1=1} & \quad & \texttt{r1=0} \end{matrix}}_{\text{SC}}$$

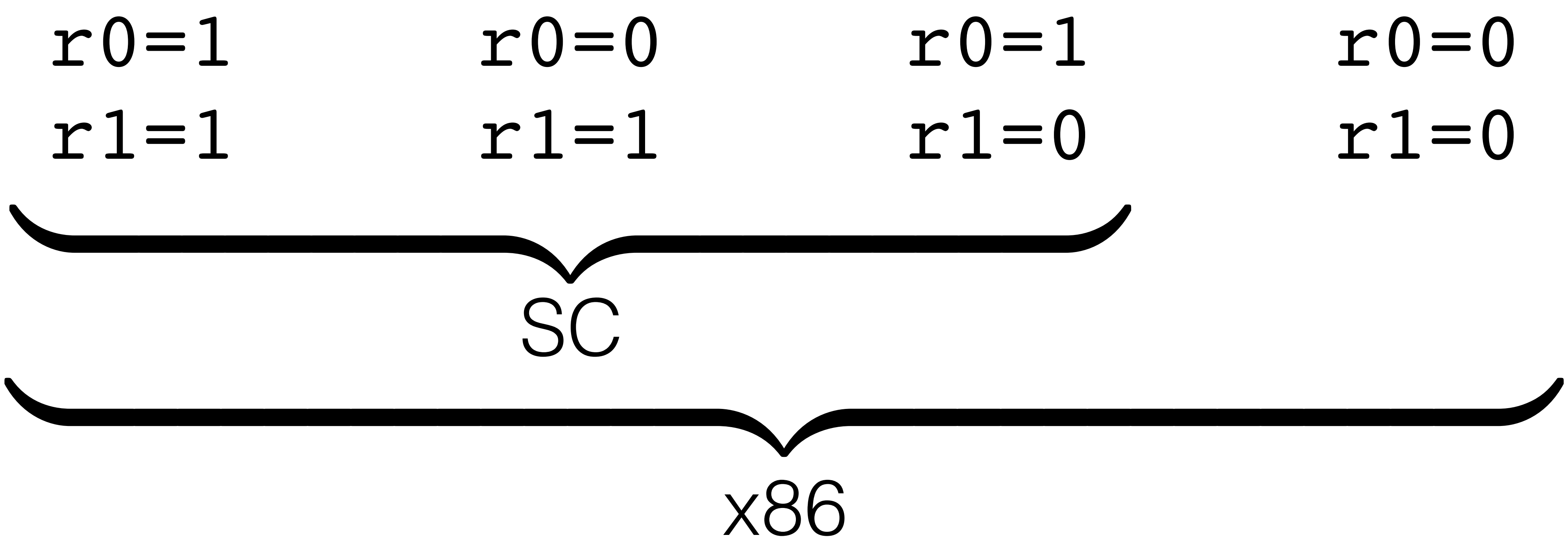
WEAK MEMORY

```
MOV [x] 1     ||  MOV [y] 1

MOV r0 [y]    ||  MOV r1 [x]
```

$$\underbrace{\begin{matrix} \texttt{r0=1} & \quad \texttt{r0=0} & \quad \texttt{r0=1} \\ \texttt{r1=1} & \quad \texttt{r1=1} & \quad \texttt{r1=0} \end{matrix}}_{\text{SC}} \qquad \begin{matrix} \texttt{r0=0} \\ \texttt{r1=0} \end{matrix}$$

WEAK MEMORY

```
MOV [x] 1    ||  MOV [y] 1
MOV r0 [y]   ||  MOV r1 [x]
```


r0=1
r1=1

r0=0
r1=1

r0=1
r1=0

r0=0
r1=0

SC

x86

WEAK MEMORY

```
MOV [x] 1    ||  MOV [y] 1

MOV r0 [y]   ||  MOV r1 [x]
```

r0=1
r1=1
	r0=0
	r1=1
		r0=1
		r1=0
			r0=0
			r1=0

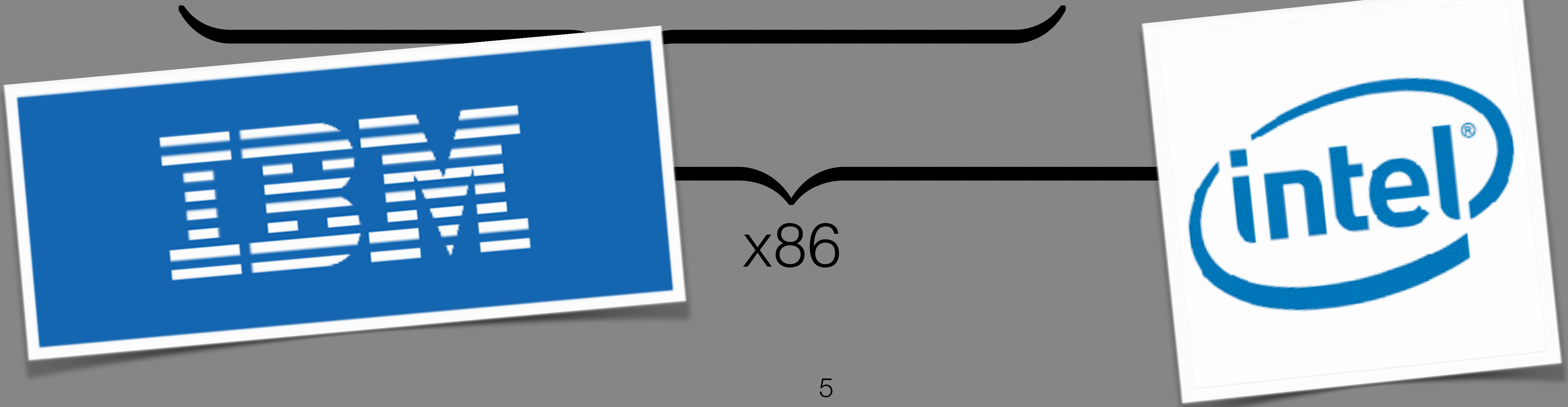
SC

x86

WEAK MEMORY

```
MOV [x] 1   ||  MOV [y] 1

MOV r0 [y]  ||  MOV r1 [x]
```

r0=1	r0=0	r0=1	r0=0
r1=1	r1=1	r1=0	r1=0

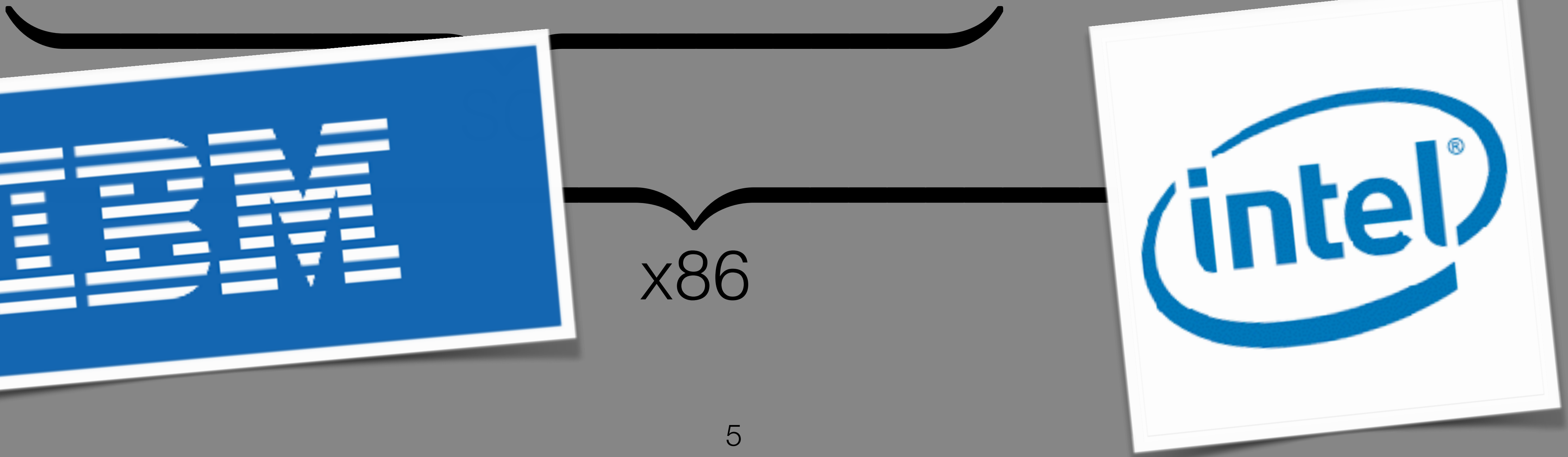


x86

AK MEMORY

```
MOV [x] 1   ||  MOV [y] 1

MOV r0 [y]  ||  MOV r1 [x]
```

```
r0=1    r0=0    r0=1    r0=0
r1=1    r1=1    r1=0    r1=0
```



AK MEMO

ARM®

NVIDIA®

```
MOV [x] 1    ||  MOV [y] 1

MOV r0 [y]   ||  MOV r1 [x]
```

r0=1	r0=0	r0=1	r0=0
r1=1	r1=1	r1=0	r1=0

IBM

x86

intel®

ARM®

AMD

NVIDIA®

```
MOV [x] 1    ‖ MOV [y] 1

MOV r0 [y]   ‖ MOV r1 [x]
```

r0=1	r0=0	r0=1	r0=0
r1=1	r1=1	r1=0	r1=0

IBM

x86

intel®

ARM®

AMD

NVIDIA®

```
MOV [x] 1   ||
MOV r0 [y]  || MOV
```

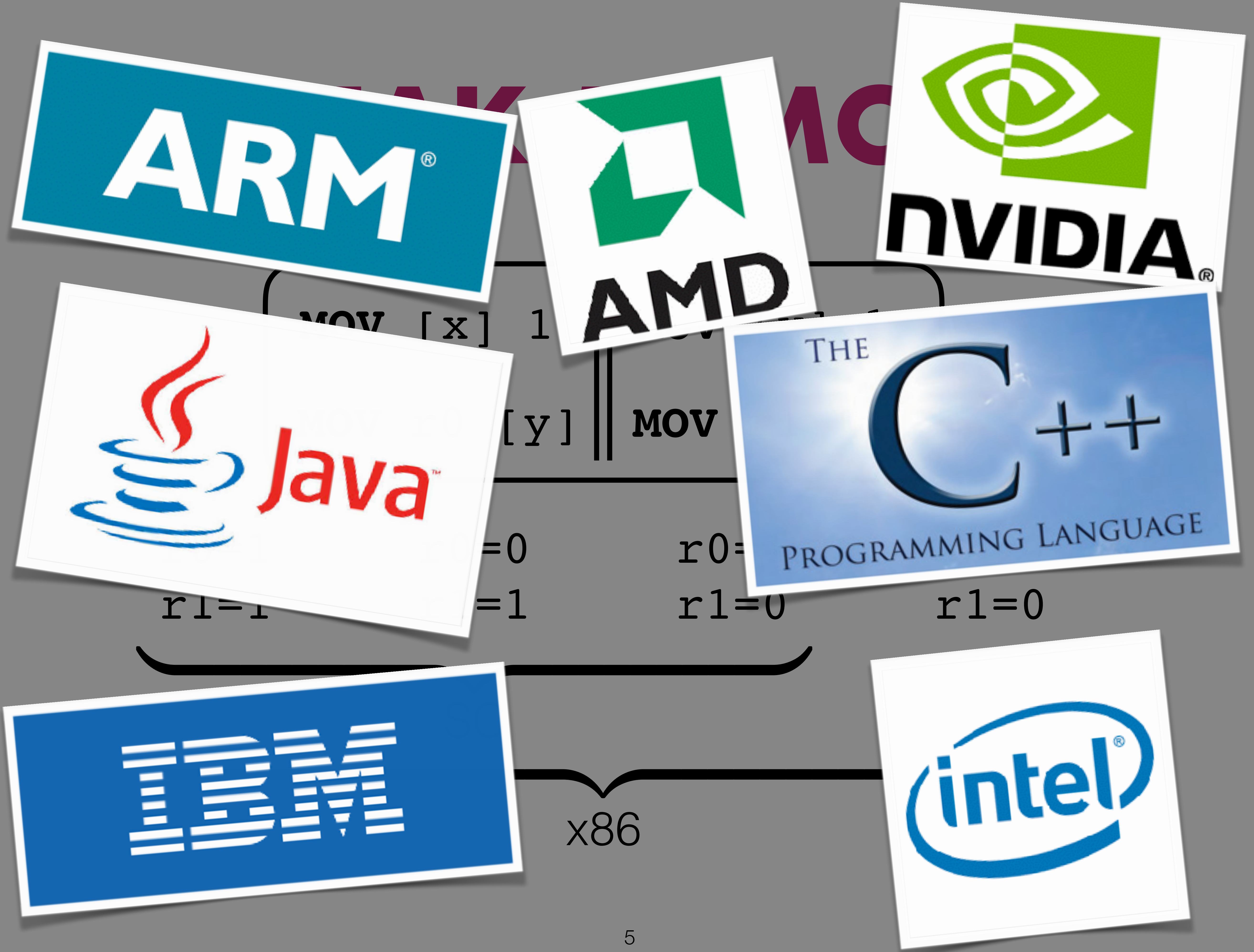
THE C++ PROGRAMMING LANGUAGE

```
r0=1      r0=0      r0=
r1=1      r1=1      r1=0      r1=0
```

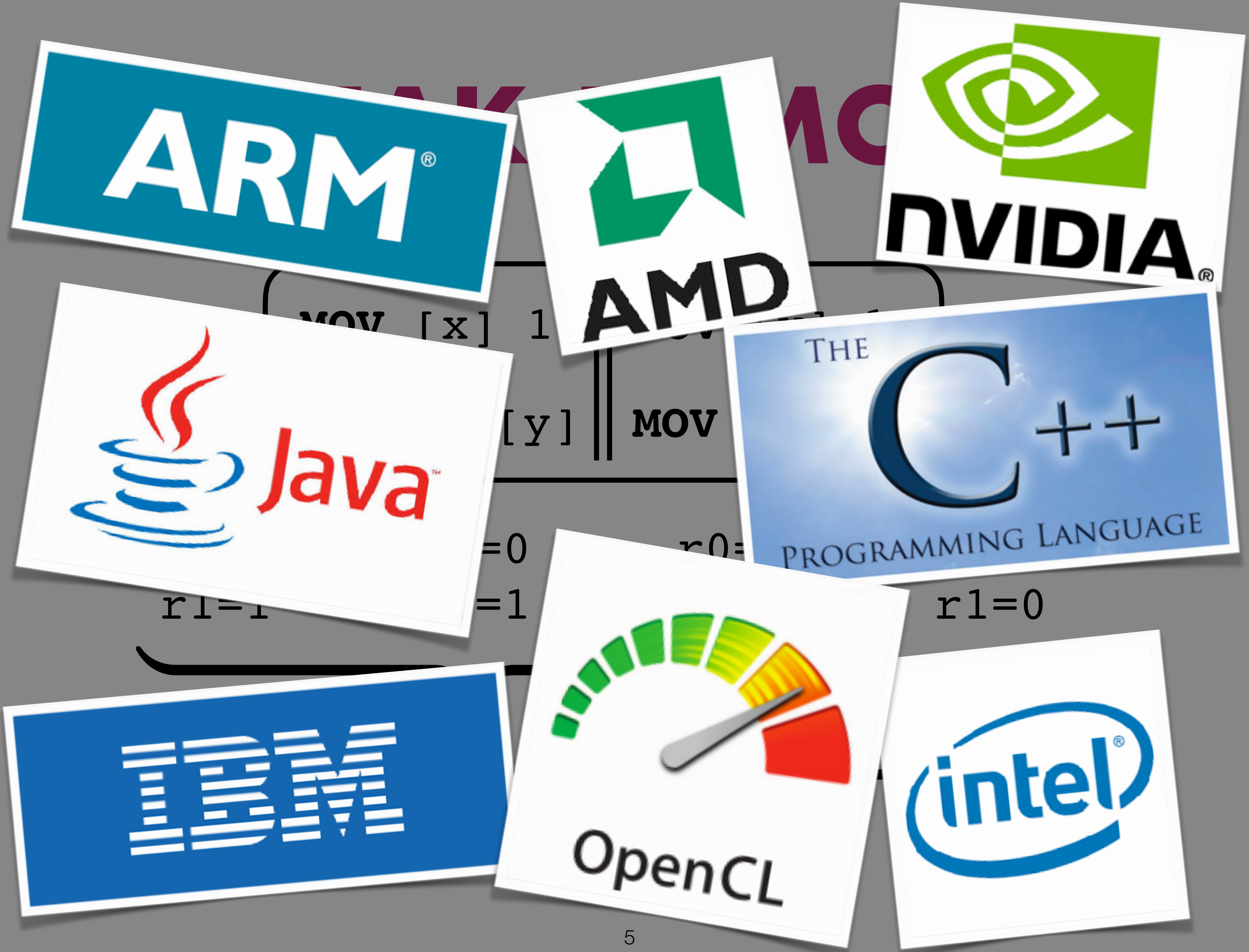
IBM

x86

intel®



```
MOV [x] 1
        [y]   MOV
      =0       r0=
r1=1  =1       r1=0    r1=0
```



MOV [x] 1

[y]

MOV

=0

r0=

r1=1

=1

r1=0

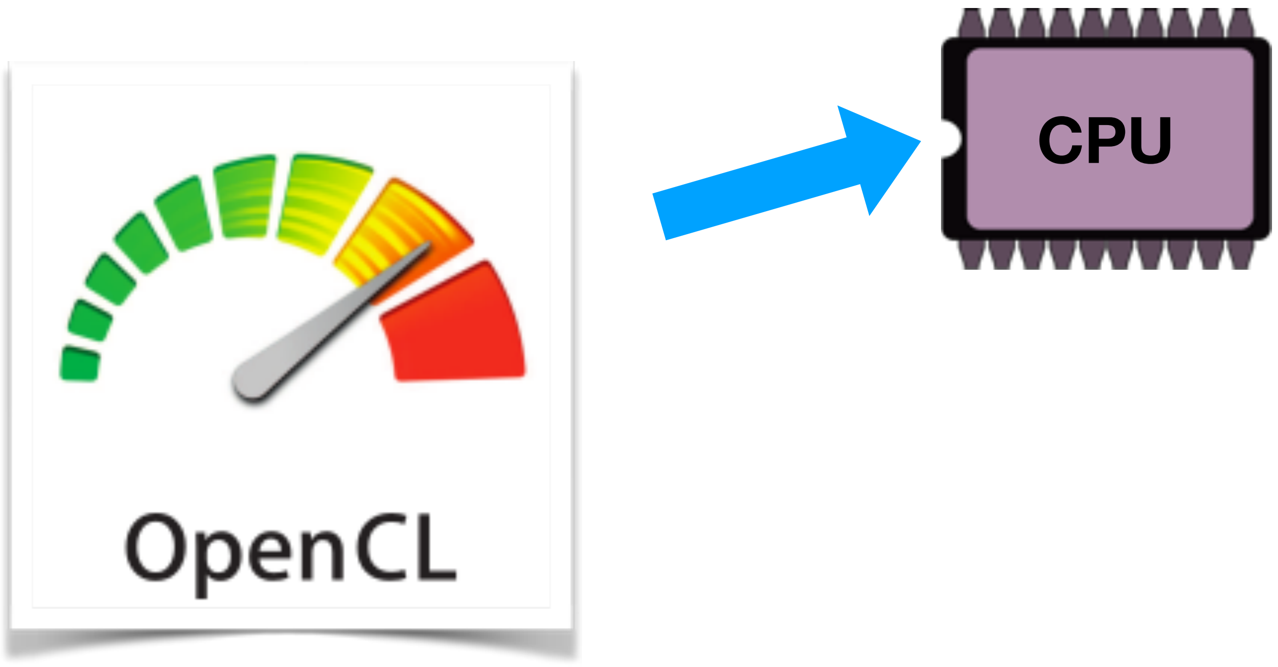
WEAK MEMORY IS HARD!

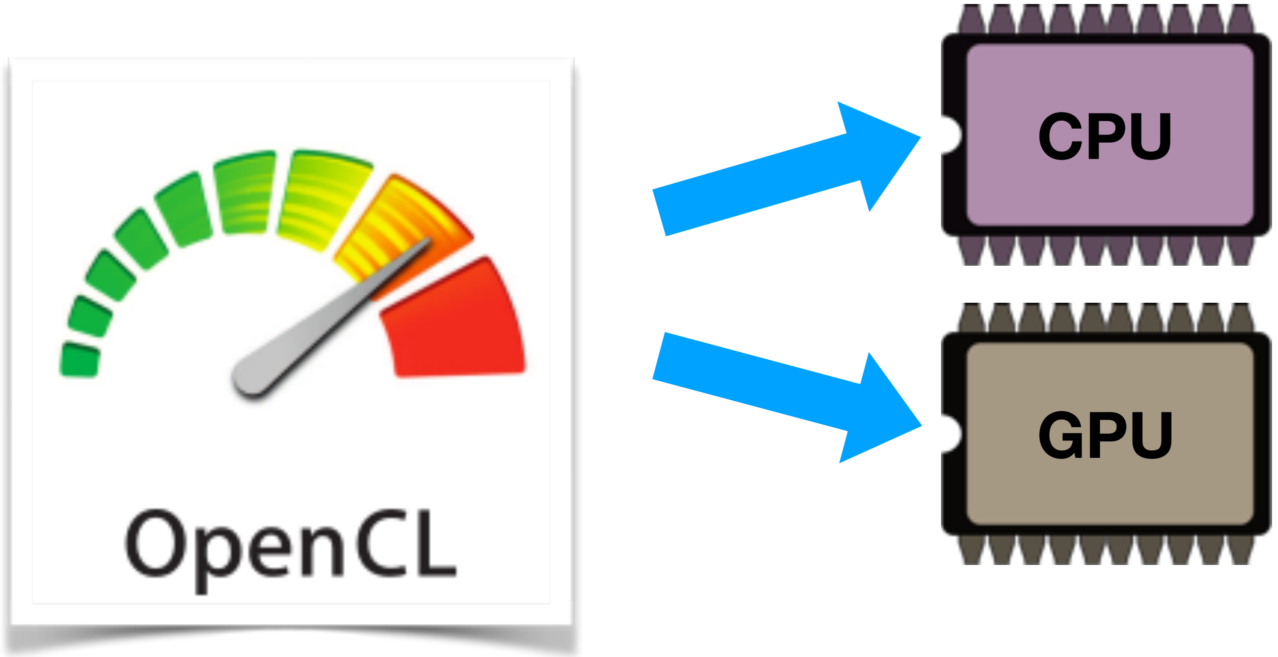
- x86 proved tricky to formalise correctly *[Sarkar et al., POPL'09; Owens et al., TPHOLs'09]*
- Bug found in deployed "Power 5" processors *[Alglave et al., CAV'10]*
- C++ specification did not guarantee its own key property *[Batty et al., POPL'11]*
- Routine compiler optimisations are invalid under Java and C++ memory models *[Sevcik, PLDI'11; Vafeiadis et al. POPL'15]*
- Behaviour of NVIDIA graphics processors contradicted NVIDIA's programming guide *[Alglave et al., ASPLOS'15]*

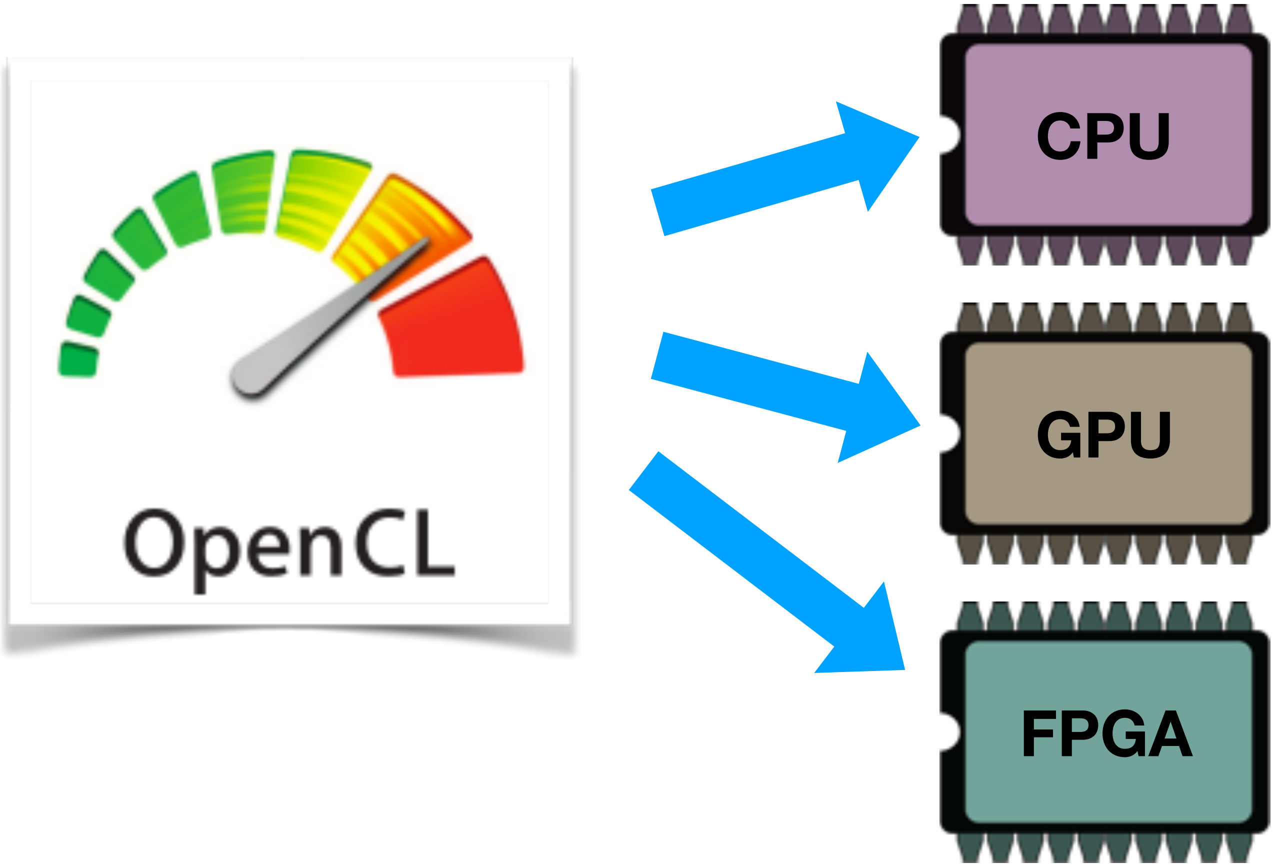
THIS TALK

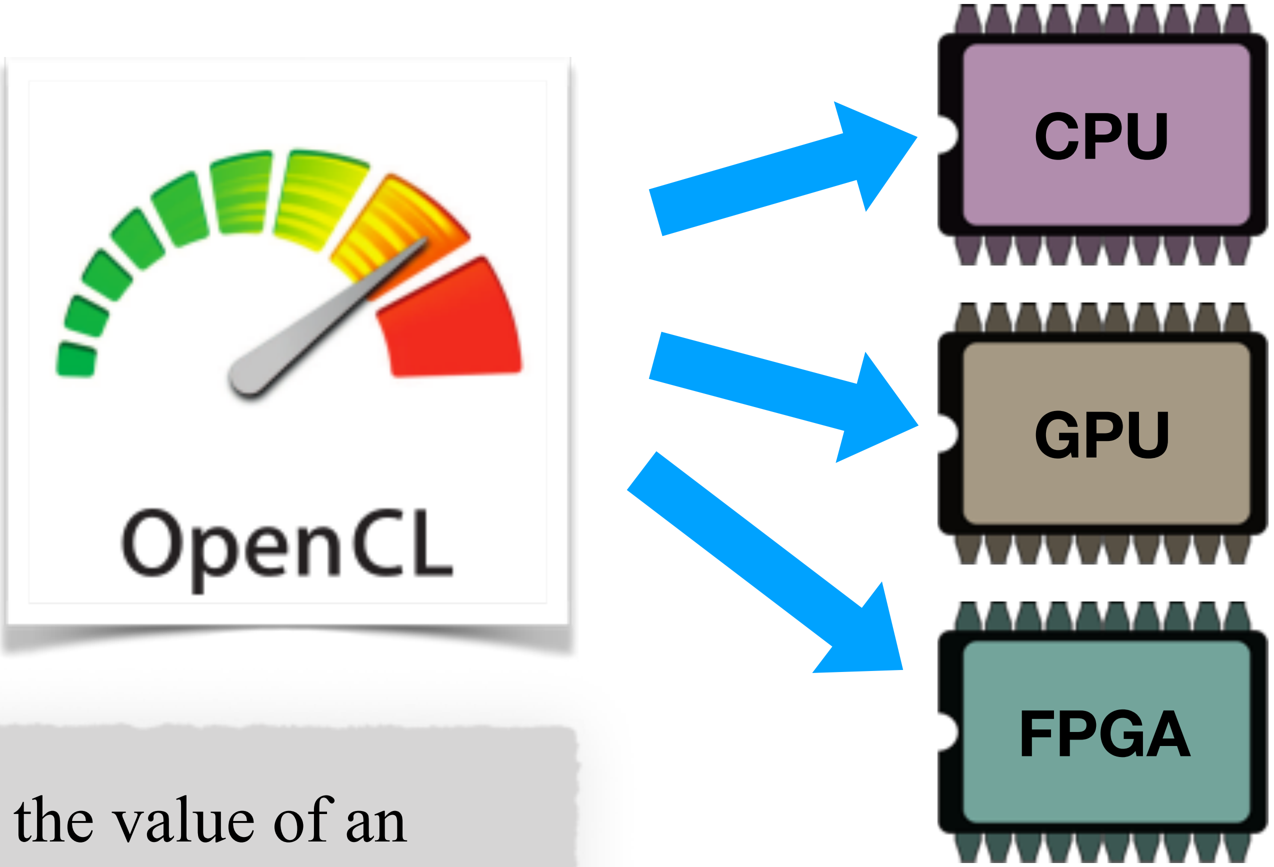
- ~~Weak memory~~
- Formalising the OpenCL memory model
- Implementing the OpenCL memory model on GPUs
- Implementing the OpenCL memory model on FPGAs

OpenCL

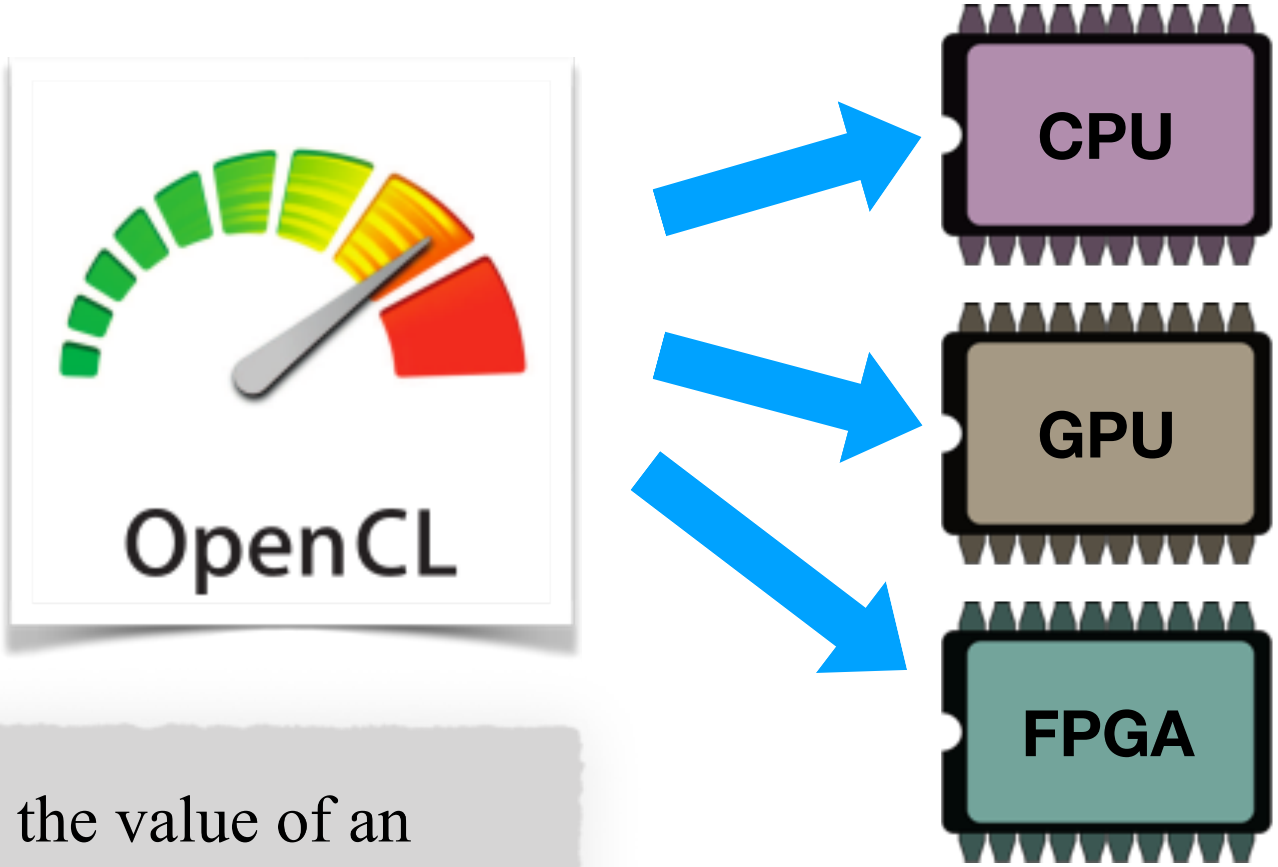

OpenCL
CPU


OpenCL
CPU
GPU


OpenCL
CPU
GPU
FPGA

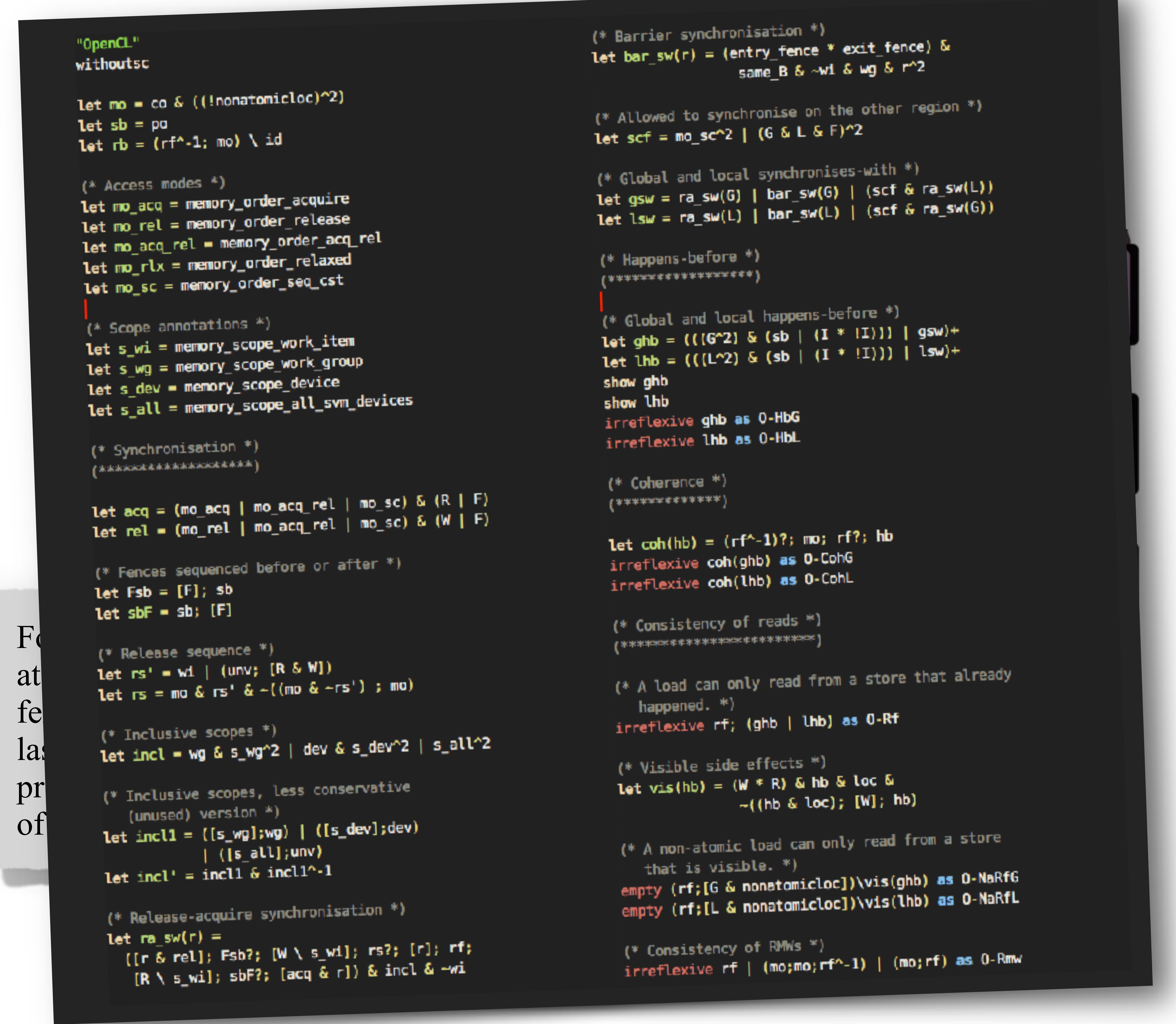


For an atomic operation **B** that reads the value of an atomic object **M**, if there is a *memory_order_seq_cst* fence **X** sequenced-before **B**, then **B** observes either the last *memory_order_seq_cst* modification of **M** preceding **X** in the total order **S** or a later modification of **M** in its modification order. *[OpenCL 2.0 standard, 2015]*



For an atomic operation **B** that reads the value of an atomic object **M**, if there is a *memory_order_seq_cst* fence **X** sequenced-before **B**, then **B** observes either the last *memory_order_seq_cst* modification of **M** preceding **X** in the total order **S** or a later modification of **M** in its modification order. *[OpenCL 2.0 standard, 2015]*

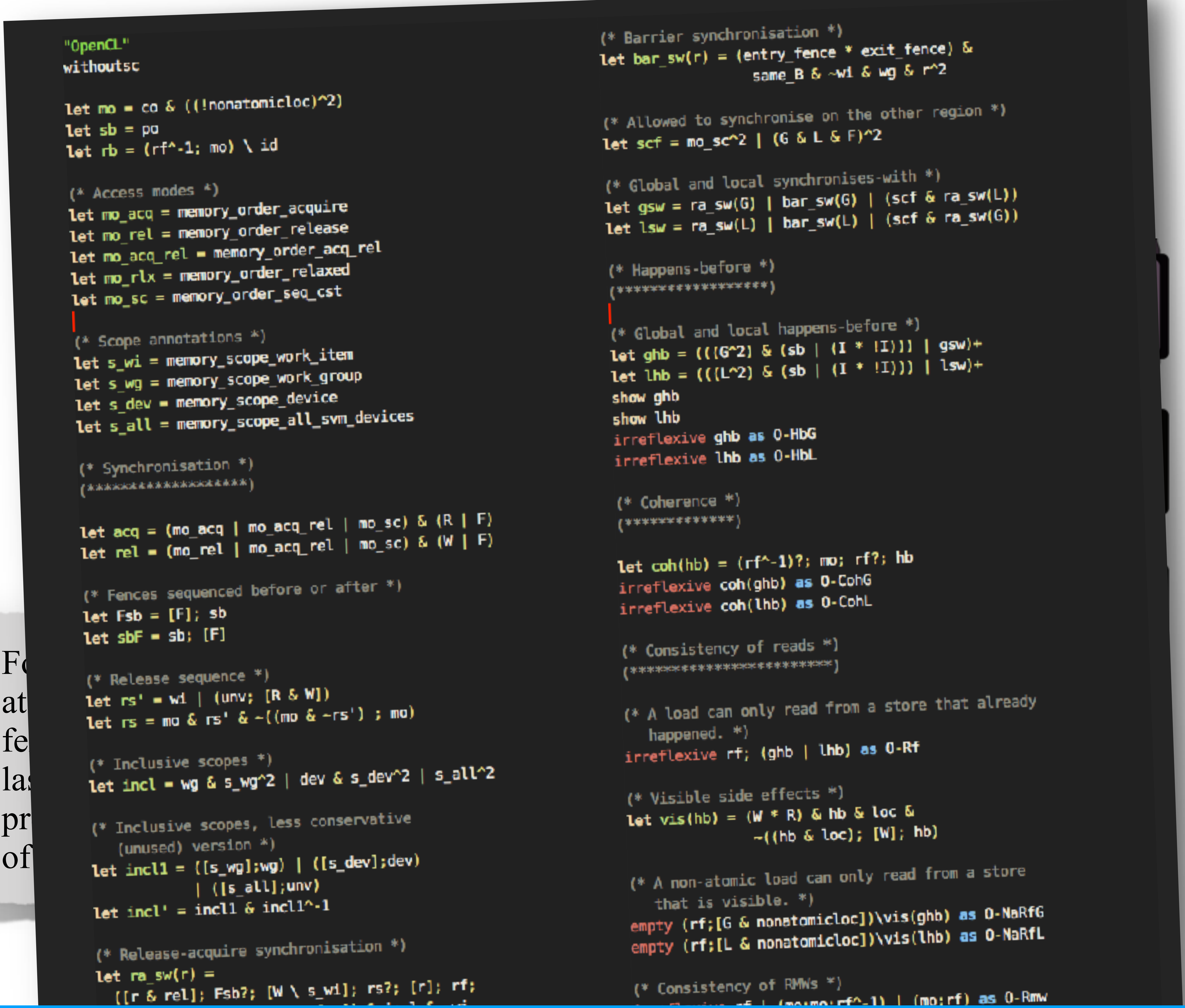
Fo
at
fe
las
pr
of



```
"OpenCL"
withoutsc

let mo = co & ((!nonatomicloc)^2)
let sb = po
let rb = (rf^-1; mo) \ id

(* Access modes *)
let mo_acq = memory_order_acquire
let mo_rel = memory_order_release
let mo_acq_rel = memory_order_acq_rel
let mo_rlx = memory_order_relaxed
let mo_sc = memory_order_seq_cst

(* Scope annotations *)
let s_wi = memory_scope_work_item
let s_wg = memory_scope_work_group
let s_dev = memory_scope_device
let s_all = memory_scope_all_svm_devices

(* Synchronisation *)
(*******************)

let acq = (mo_acq | mo_acq_rel | mo_sc) & (R | F)
let rel = (mo_rel | mo_acq_rel | mo_sc) & (W | F)

(* Fences sequenced before or after *)
let Fsb = [F]; sb
let sbF = sb; [F]

(* Release sequence *)
let rs' = wi | (unv; [R & W])
let rs = mo & rs' & ~((mo & ~rs') ; mo)

(* Inclusive scopes *)
let incl = wg & s_wg^2 | dev & s_dev^2 | s_all^2

(* Inclusive scopes, less conservative
   (unused) version *)
let incl1 = ([s_wg];wg) | ([s_dev];dev)
              | ([s_all];unv)
let incl' = incl1 & incl1^-1

(* Release-acquire synchronisation *)
let ra_sw(r) =
  ([r & rel]; Fsb?; [W \ s_wi]; rs?; [r]; rf;
   [R \ s_wi]; sbF?; [acq & r]) & incl & ~wi
```

```
(* Barrier synchronisation *)
let bar_sw(r) = (entry_fence * exit_fence) &
                  same_B & ~wi & wg & r^2

(* Allowed to synchronise on the other region *)
let scf = mo_sc^2 | (G & L & F)^2

(* Global and local synchronises-with *)
let gsw = ra_sw(G) | bar_sw(G) | (scf & ra_sw(L))
let lsw = ra_sw(L) | bar_sw(L) | (scf & ra_sw(G))

(* Happens-before *)
(******************)

(* Global and local happens-before *)
let ghb = (((G^2) & (sb | (I * !I))) | gsw)+
let lhb = (((L^2) & (sb | (I * !I))) | lsw)+
show ghb
show lhb
irreflexive ghb as O-HbG
irreflexive lhb as O-HbL

(* Coherence *)
(*************)

let coh(hb) = (rf^-1)?; mo; rf?; hb
irreflexive coh(ghb) as O-CohG
irreflexive coh(lhb) as O-CohL

(* Consistency of reads *)
(************************)

(* A load can only read from a store that already
   happened. *)
irreflexive rf; (ghb | lhb) as O-Rf

(* Visible side effects *)
let vis(hb) = (W * R) & hb & loc &
                ~((hb & loc); [W]; hb)

(* A non-atomic load can only read from a store
   that is visible. *)
empty (rf;[G & nonatomicloc])\vis(ghb) as O-NaRfG
empty (rf;[L & nonatomicloc])\vis(lhb) as O-NaRfL

(* Consistency of RMWs *)
irreflexive rf | (mo;mo;rf^-1) | (mo;rf) as O-Rmw
```

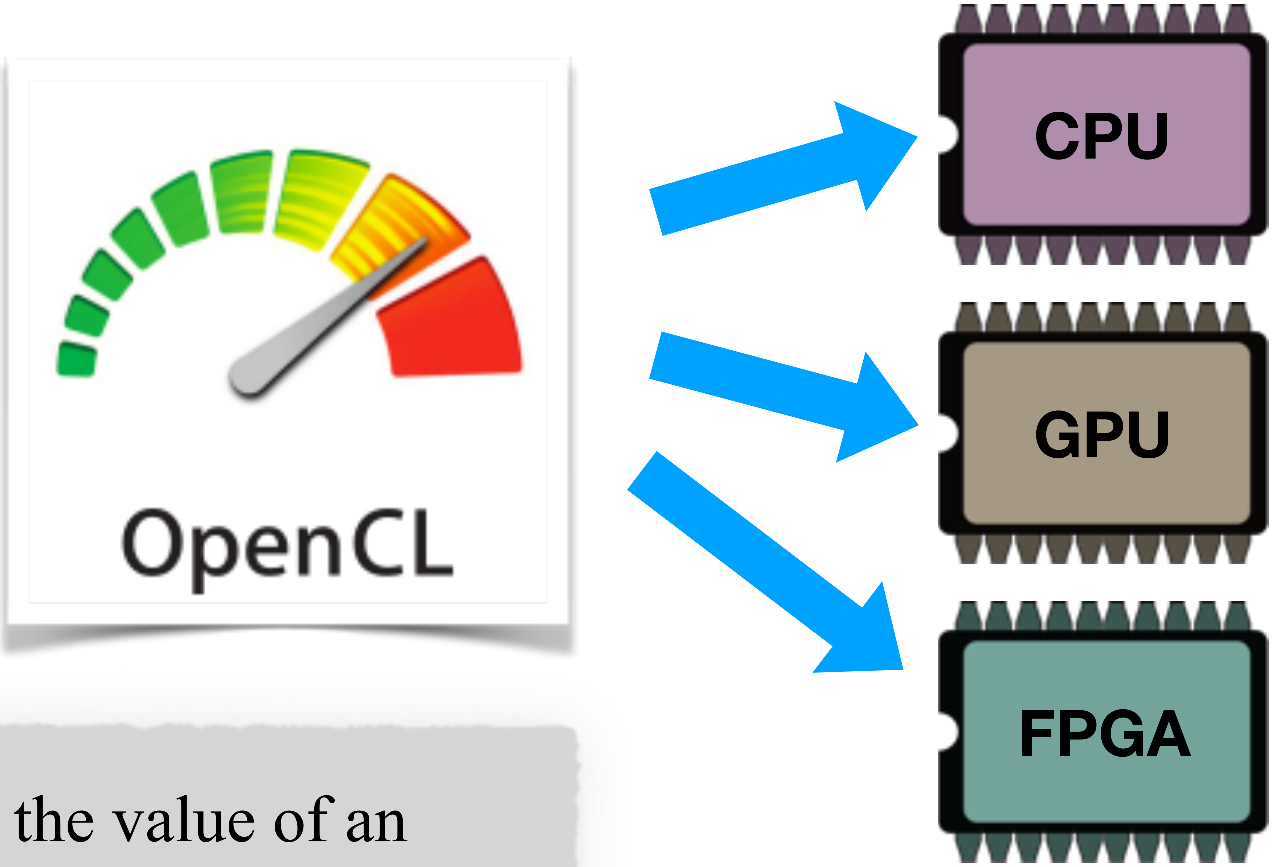
Fo
at
fe
la
pr
of



```
"OpenCL"
withoutsc

let mo = co & ((!nonatomicloc)^2)
let sb = po
let rb = (rf^-1; mo) \ id

(* Access modes *)
let mo_acq = memory_order_acquire
let mo_rel = memory_order_release
let mo_acq_rel = memory_order_acq_rel
let mo_rlx = memory_order_relaxed
let mo_sc = memory_order_seq_cst

(* Scope annotations *)
let s_wi = memory_scope_work_item
let s_wg = memory_scope_work_group
let s_dev = memory_scope_device
let s_all = memory_scope_all_svm_devices

(* Synchronisation *)
(*****************)

let acq = (mo_acq | mo_acq_rel | mo_sc) & (R | F)
let rel = (mo_rel | mo_acq_rel | mo_sc) & (W | F)

(* Fences sequenced before or after *)
let Fsb = [F]; sb
let sbF = sb; [F]

(* Release sequence *)
let rs' = wi | (unv; [R & W])
let rs = mo & rs' & ~((mo & ~rs') ; mo)

(* Inclusive scopes *)
let incl = wg & s_wg^2 | dev & s_dev^2 | s_all^2

(* Inclusive scopes, less conservative
   (unused) version *)
let incl1 = ([s_wg];wg) | ([s_dev];dev)
            | ([s_all];unv)
let incl' = incl1 & incl1^-1

(* Release-acquire synchronisation *)
let ra_sw(r) =
  ([r & rel]; Fsb?; [W \ s_wi]; rs?; [r]; rf;
```

```
(* Barrier synchronisation *)
let bar_sw(r) = (entry_fence * exit_fence) &
                 same_B & ~wi & wg & r^2

(* Allowed to synchronise on the other region *)
let scf = mo_sc^2 | (G & L & F)^2

(* Global and local synchronises-with *)
let gsw = ra_sw(G) | bar_sw(G) | (scf & ra_sw(L))
let lsw = ra_sw(L) | bar_sw(L) | (scf & ra_sw(G))

(* Happens-before *)
(*****************)

(* Global and local happens-before *)
let ghb = (((G^2) & (sb | (I * !I))) | gsw)+
let lhb = (((L^2) & (sb | (I * !I))) | lsw)+
show ghb
show lhb
irreflexive ghb as O-HbG
irreflexive lhb as O-HbL

(* Coherence *)
(*************)

let coh(hb) = (rf^-1)?; mo; rf?; hb
irreflexive coh(ghb) as O-CohG
irreflexive coh(lhb) as O-CohL

(* Consistency of reads *)
(************************)

(* A load can only read from a store that already
   happened. *)
irreflexive rf; (ghb | lhb) as O-Rf

(* Visible side effects *)
let vis(hb) = (W * R) & hb & loc &
              ~((hb & loc); [W]; hb)

(* A non-atomic load can only read from a store
   that is visible. *)
empty (rf;[G & nonatomicloc])\vis(ghb) as O-NaRfG
empty (rf;[L & nonatomicloc])\vis(lhb) as O-NaRfL

(* Consistency of RMWs *)
irreflexive rf | (mo;mo;rf^-1) | (mo;rf) as O-Rmw
```

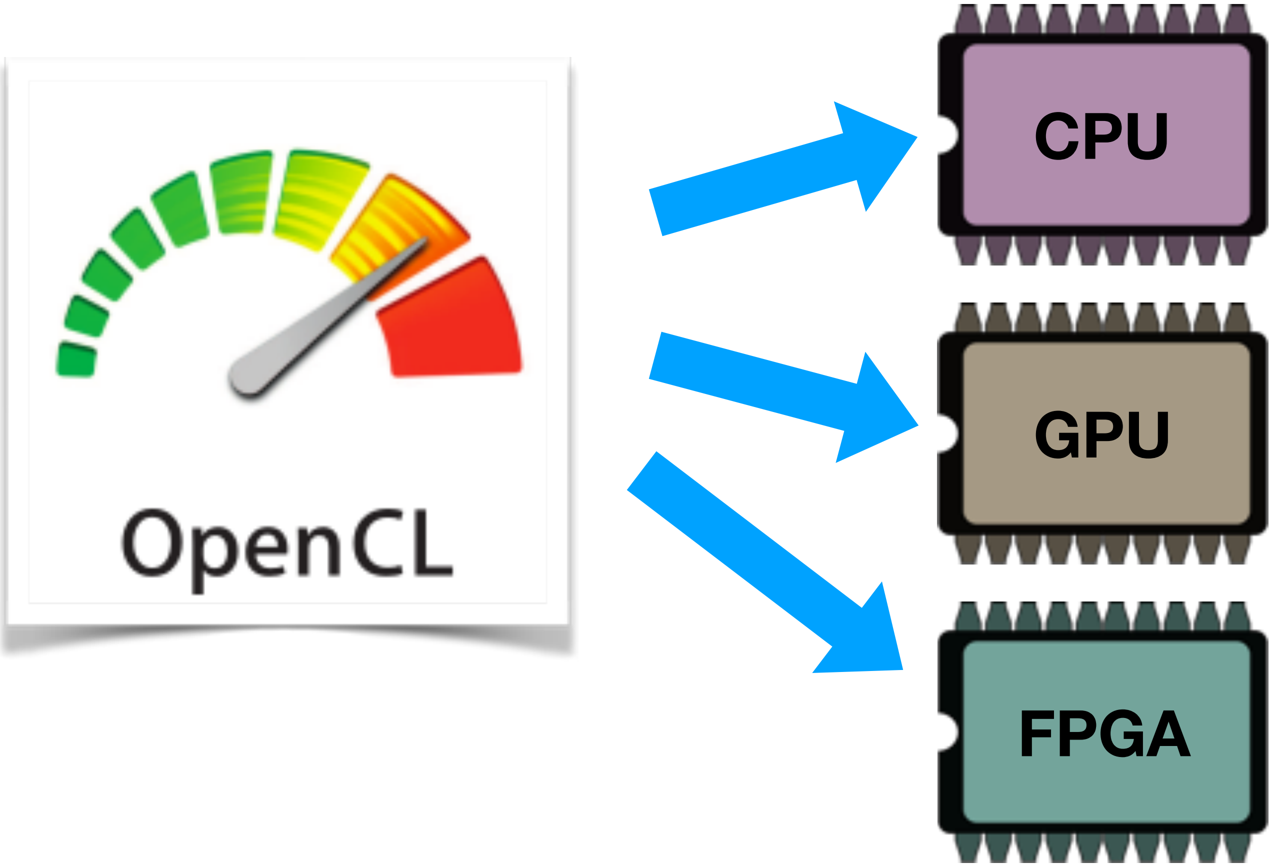
Key publication: ACM **POPL** 2016 (with Batty & Donaldson)

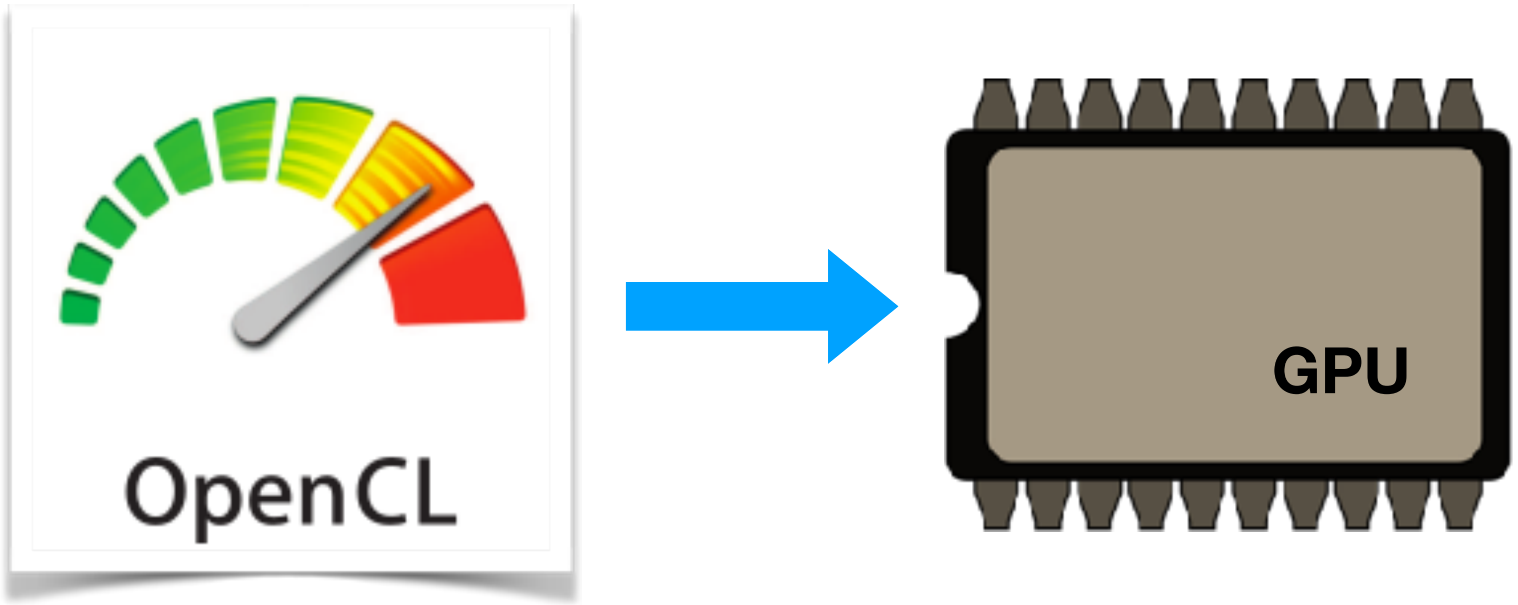


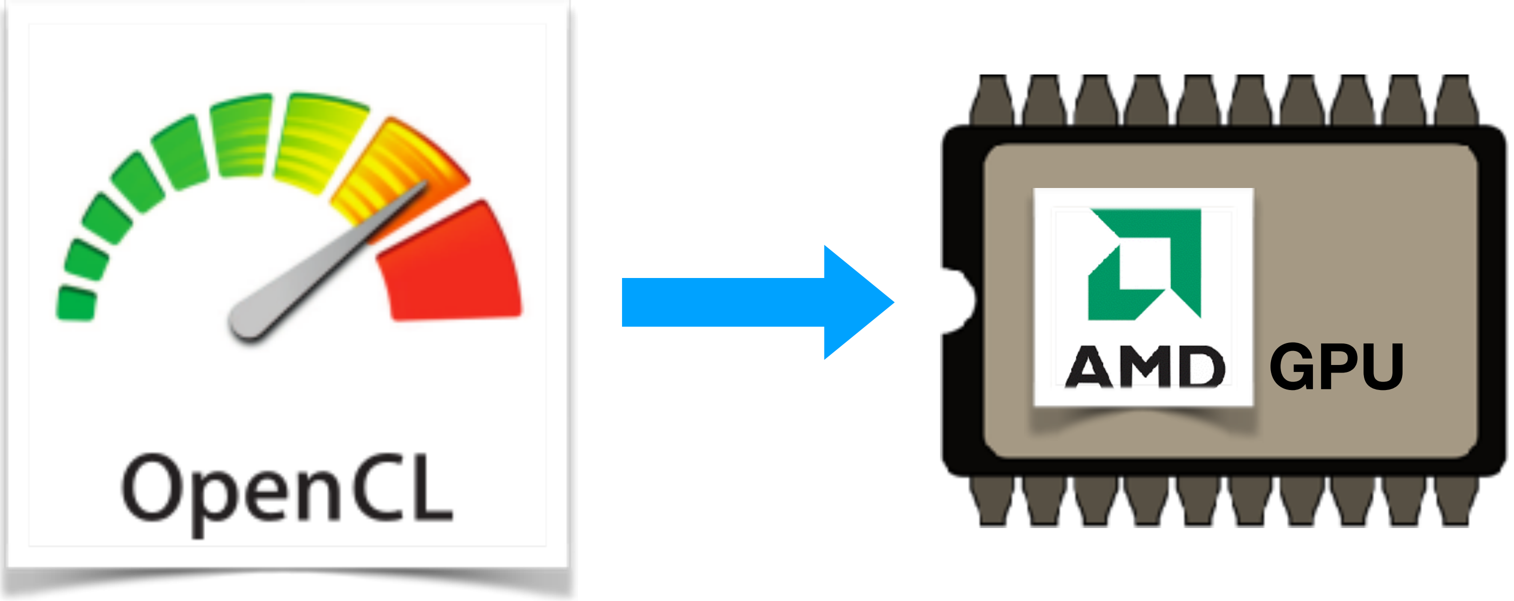
For an atomic operation **B** that reads the value of an atomic object **M**, if there is a *memory_order_seq_cst* fence **X** sequenced-before **B**, then **B** observes either the last *memory_order_seq_cst* modification of **M** preceding **X** in the total order **S** or a later modification of **M** in its modification order. *[OpenCL 2.0 standard, 2015]*

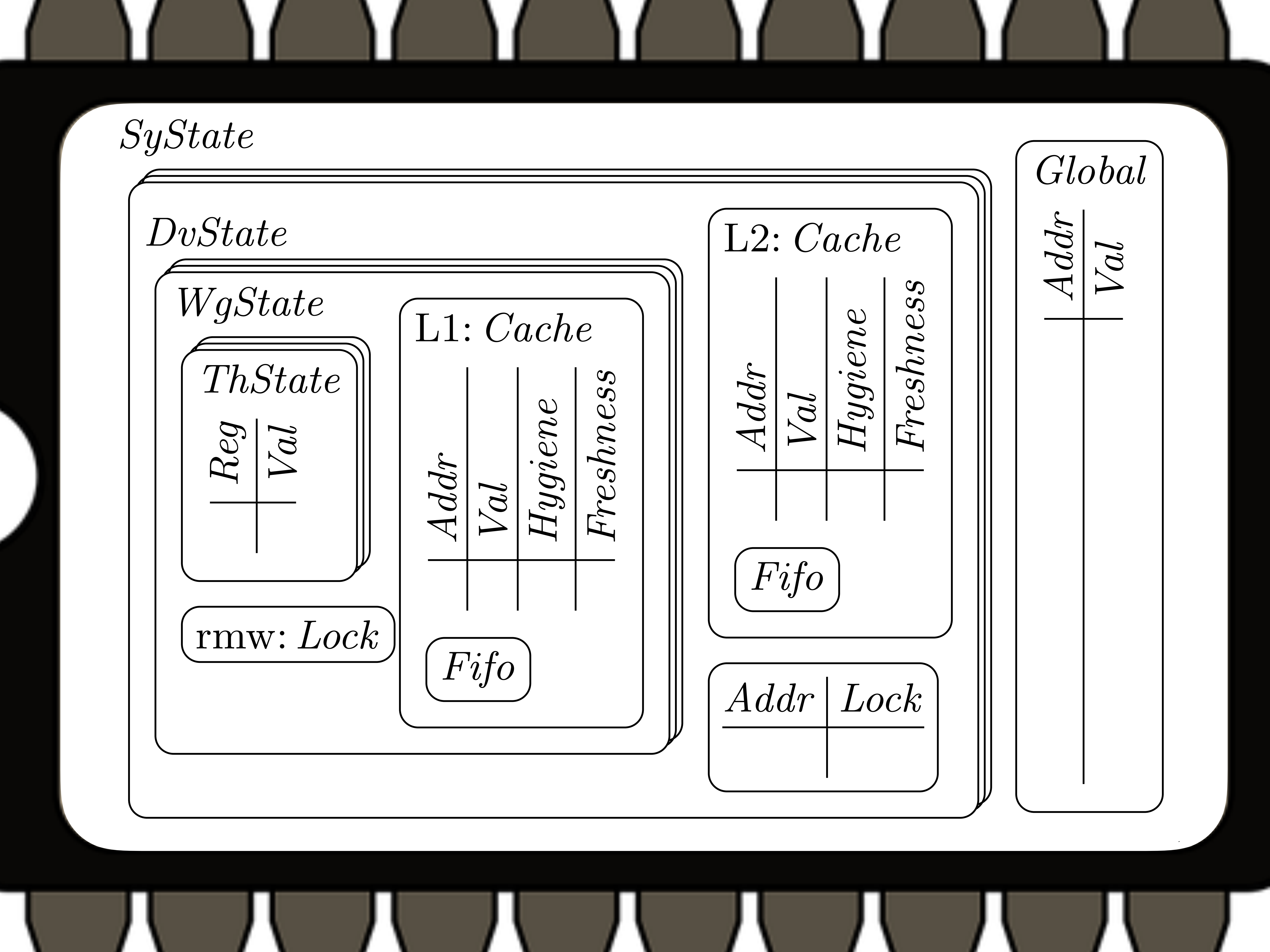
THIS TALK

- ~~Weak memory~~
- ~~Formalising the OpenCL memory model~~
- Implementing the OpenCL memory model on GPUs
- Implementing the OpenCL memory model on FPGAs

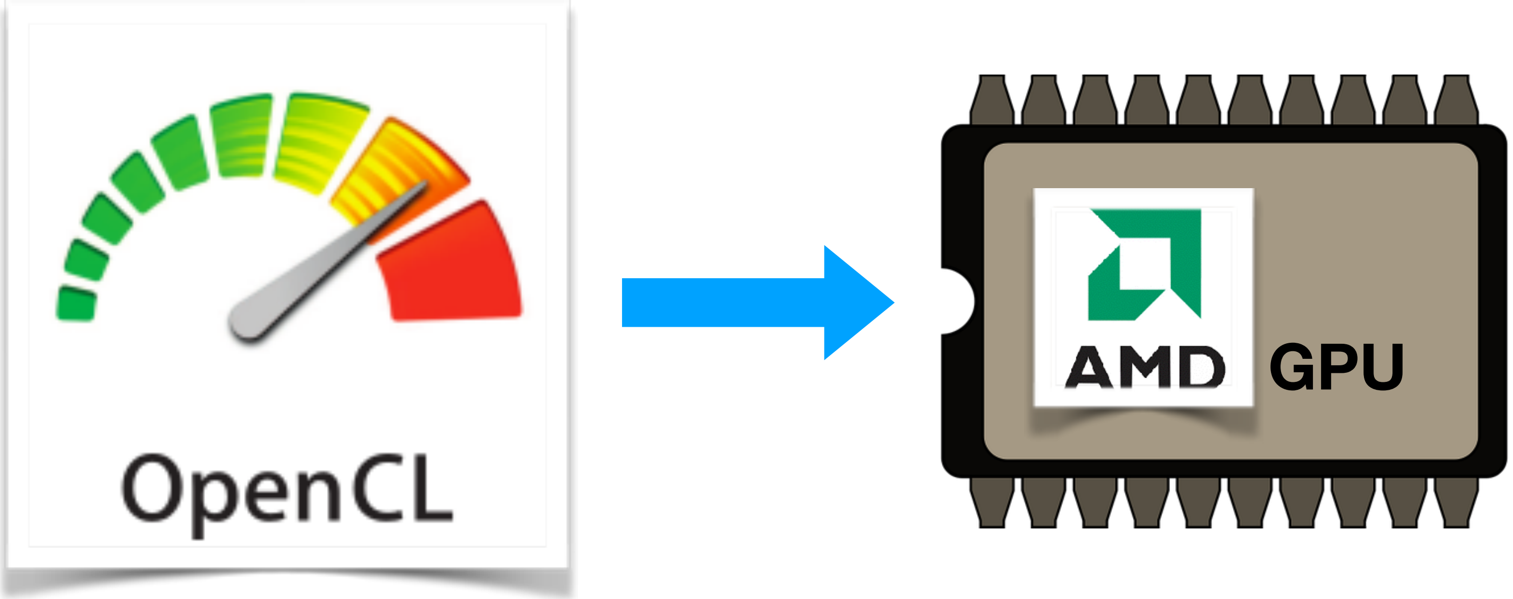

OpenCL
CPU
GPU
FPGA


OpenCL
GPU

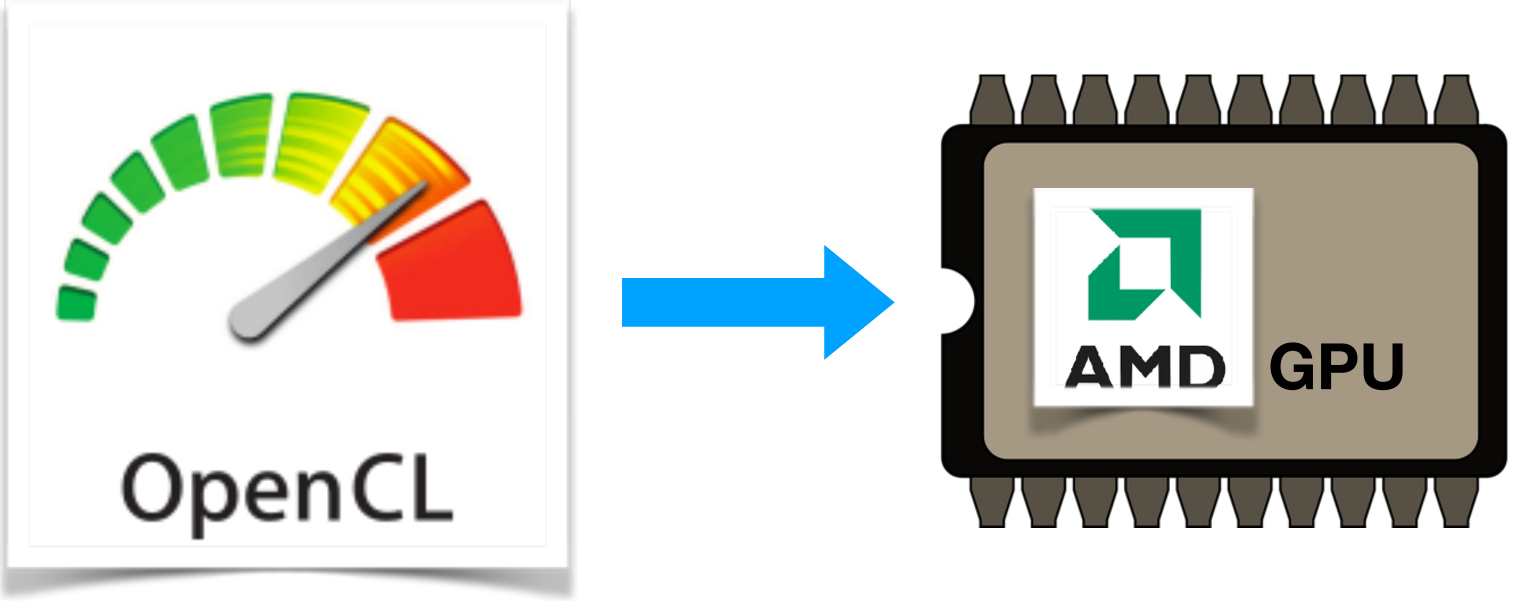

OpenCL
AMD
GPU



SyState
DvState
WgState
ThState
Reg
Val
rmw: Lock
L1: Cache
Addr
Val
Hygiene
Freshness
Fifo
L2: Cache
Addr
Val
Hygiene
Freshness
Fifo
Addr
Lock
Global
Addr
Val


OpenCL
AMD
GPU

	na or WG	DV (not remote)	DV (remote)
r=load(x)	LD r x	INV$_{L1}$ WG LD r x	$\left.\begin{array}{l}\text{FLU}_{L1}\ \text{DV}\\ \text{INV}_{L1}\ \text{WG}\\ \text{LD r x}\end{array}\right\}$ LK x
store(x,r)	ST r x	FLU$_{L1}$ WG ST r x	$\left.\begin{array}{l}\text{FLU}_{L1}\ \text{WG}\\ \text{ST r x}\\ \text{INV}_{L1}\ \text{DV}\end{array}\right\}$ LK x
r=fetch_inc(x)	INC$_{L1}$ r x	FLU$_{L1}$ WG INV$_{L1}$ WG INC$_{L2}$ r x	$\left.\begin{array}{l}\text{FLU}_{L1}\ \text{DV}\\ \text{INV}_{L1}\ \text{WG}\\ \text{INC}_{L2}\ \text{r x}\\ \text{INV}_{L1}\ \text{DV}\end{array}\right\}$ LK x LK$_{rmw}$


OpenCL
AMD
GPU

	na or WG	DV (not remote)	DV (remote)
r=load(x)	LD r x	INV$_{L1}$ WG LD r x	$\left.\begin{array}{l}\text{FLU}_{\text{L1}}\ \text{DV}\\ \text{INV}_{\text{L1}}\ \text{WG}\\ \text{LD r x}\end{array}\right\}$ LK x
store(x,r)	ST r x	FLU$_{L1}$ WG ST r x	$\left.\begin{array}{l}\text{FLU}_{\text{L1}}\ \text{WG}\\ \text{ST r x}\\ \text{INV}_{\text{L1}}\ \text{DV}\end{array}\right\}$ LK x
r=fetch_inc(x)	INC$_{L1}$ r x	FLU$_{L1}$ WG INV$_{L1}$ WG INC$_{L2}$ r x	$\left.\begin{array}{l}\text{FLU}_{\text{L1}}\ \text{DV}\\ \text{INV}_{\text{L1}}\ \text{WG}\\ \text{INC}_{\text{L2}}\ \text{r x}\\ \text{INV}_{\text{L1}}\ \text{DV}\end{array}\right\}$ LK x LK$_{rmw}$

	na or WG	DV (not remote)	DV (remote)
r=load(x)	LD r x	INV$_{L1}$ WG LD r x	$\left.\begin{array}{l}\text{FLU}_{L1}\ \text{DV}\\ \text{INV}_{L1}\ \text{WG}\\ \text{LD r x}\end{array}\right\}$ LK x
store(x,r)	ST r x	FLU$_{L1}$ WG ST r x	$\left.\begin{array}{l}\text{FLU}_{L1}\ \text{WG}\\ \text{ST r x}\\ \text{INV}_{L1}\ \text{DV}\end{array}\right\}$ LK x
r=fetch_inc(x)	INC$_{L1}$ r x	FLU$_{L1}$ WG INV$_{L1}$ WG INC$_{L2}$ r x	$\left.\begin{array}{l}\text{FLU}_{L1}\ \text{DV}\\ \text{INV}_{L1}\ \text{WG}\\ \text{INC}_{L2}\ \text{r x}\\ \text{INV}_{L1}\ \text{DV}\end{array}\right\}$ LK x LK$_{rmw}$

message passing error

	na or WG	DV (not remote)	DV (remote)
r=load(x)	LD r x	INV$_{L1}$ WG LD r x	FLU$_{L1}$ DV INV$_{L1}$ WG LD r x LK x
store(x,r)	ST r x	FLU$_{L1}$ WG ST r x	FLU$_{L1}$ WG ST r x INV$_{L1}$ DV LK x
r=fetch_inc(x)	INC$_{L1}$ r x	FLU$_{L1}$ WG INV$_{L1}$ WG INC$_{L2}$ r x	FLU$_{L1}$ DV INV$_{L1}$ WG INC$_{L2}$ r x INV$_{L1}$ DV LK x LK$_{rmw}$

message passing error

RMW atomicity error

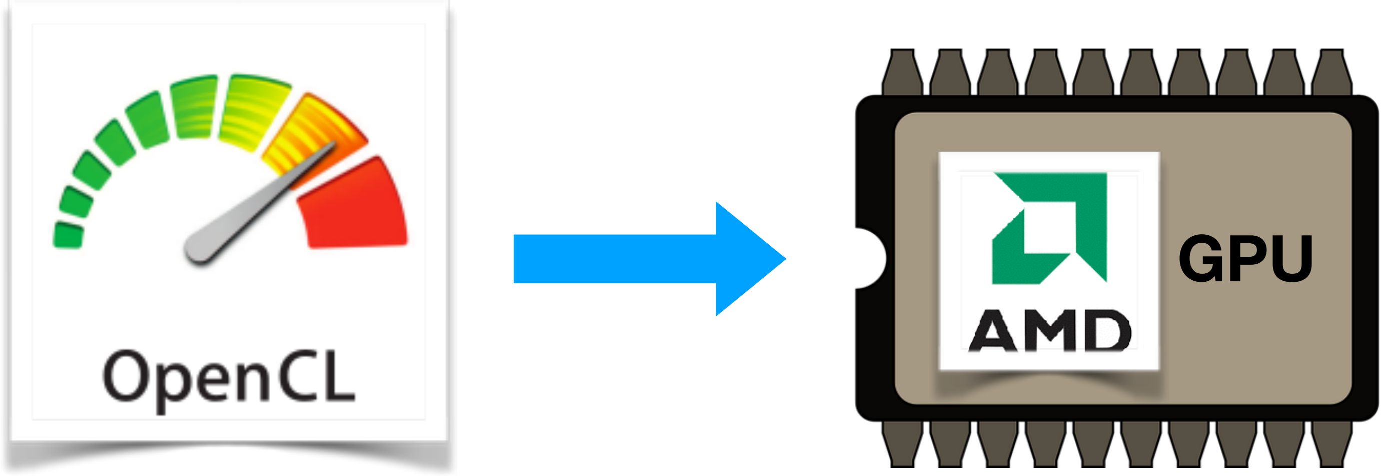
	na or WG	DV (not remote)	DV (remote)
r=load(x)	LD r x	INV$_{L1}$ WG LD r x	FLU$_{L1}$ DV INV$_{L1}$ WG LD r x LK x
store(x,r)	ST r x	FLU$_{L1}$ WG ST r x	FLU$_{L1}$ WG ST r x INV$_{L1}$ DV LK x
r=fetch_inc(x)	INC$_{L1}$ r x	FLU$_{L1}$ WG INV$_{L1}$ WG INC$_{L2}$ r x	FLU$_{L1}$ DV INV$_{L1}$ WG INC$_{L2}$ r x INV$_{L1}$ DV LK x LK$_{rmw}$

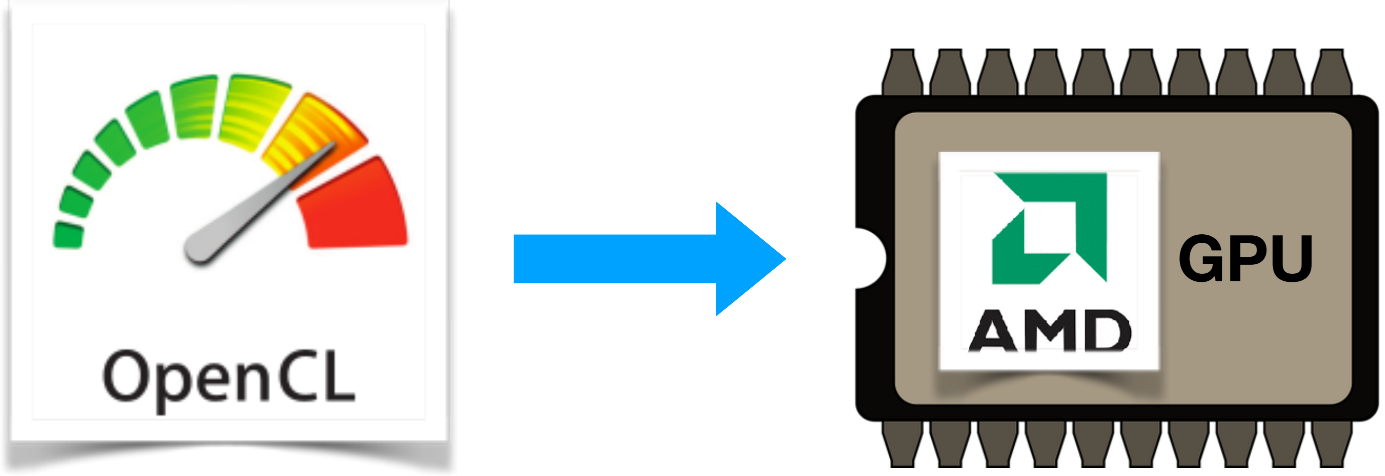
unnecessary locking

message passing error

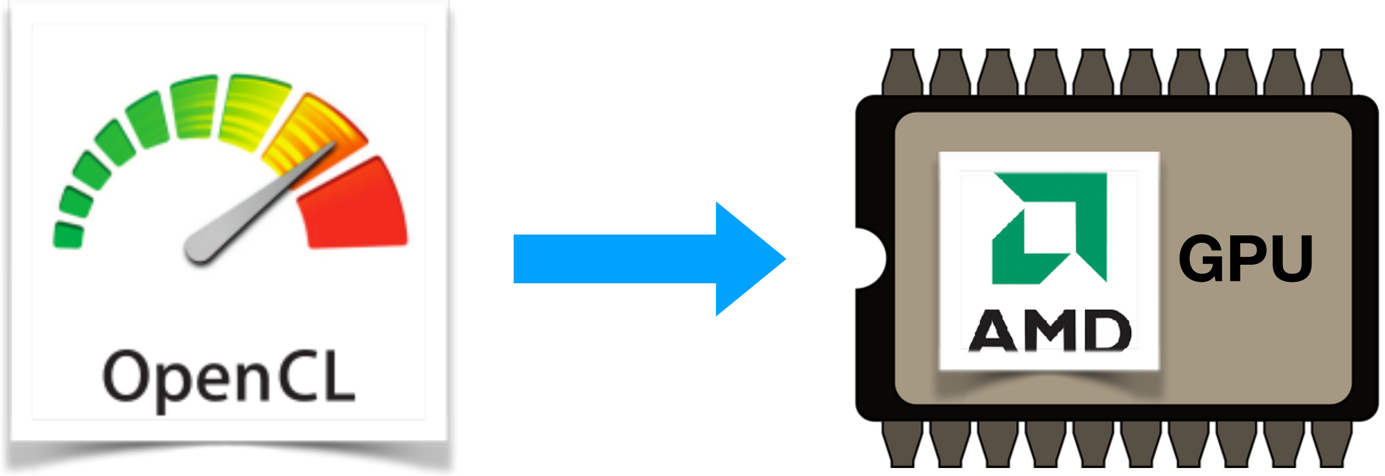
RMW atomicity error

	na or WG	DV (not remote)	DV (remote)
r=load(x)	LD r x	LD r x INV$_{L1}$ WG	LD r x FLU$_{L1}$ DV INV$_{L1}$ WG
store(x,r)	ST r x	FLU$_{L1}$ WG ST r x	$\left.\begin{array}{l}\text{FLU}_{L1}\ \text{WG}\\ \text{INV}_{L1}\ \text{DV}\\ \text{ST r x}\end{array}\right\}\ \text{LK}_{rmw}$
r=fetch_inc(x)	INC$_{L1}$ r x	FLU$_{L1}$ WG INC$_{L2}$ r x INV$_{L1}$ WG	$\left.\begin{array}{l}\text{FLU}_{L1}\ \text{WG}\\ \text{INV}_{L1}\ \text{DV}\\ \text{INC}_{L2}\ \text{r x}\\ \text{FLU}_{L1}\ \text{DV}\\ \text{INV}_{L1}\ \text{WG}\end{array}\right\}\ \text{LK}_{rmw}$


OpenCL
AMD
GPU

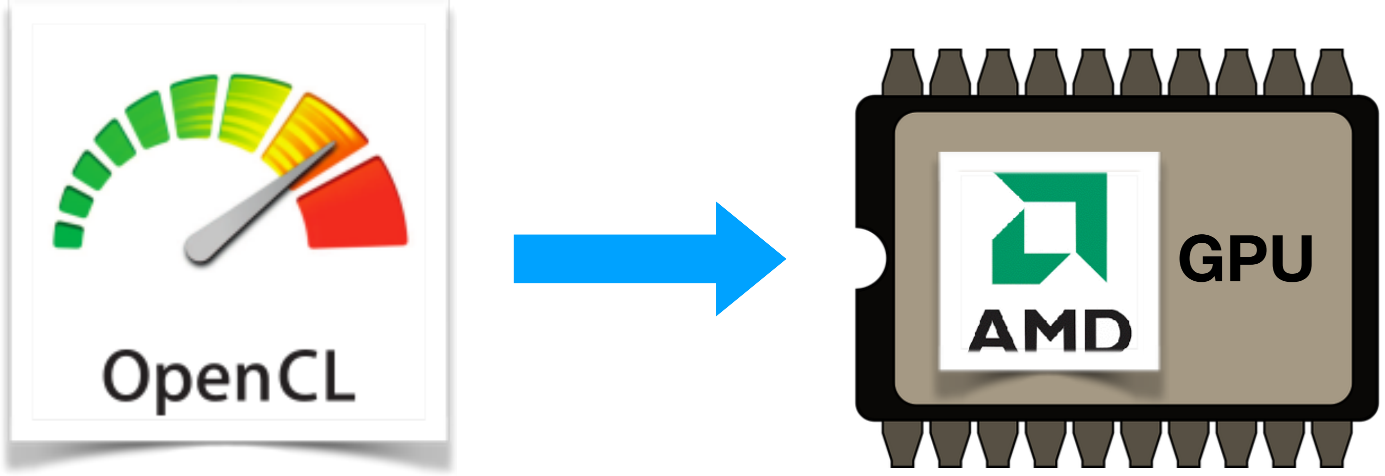


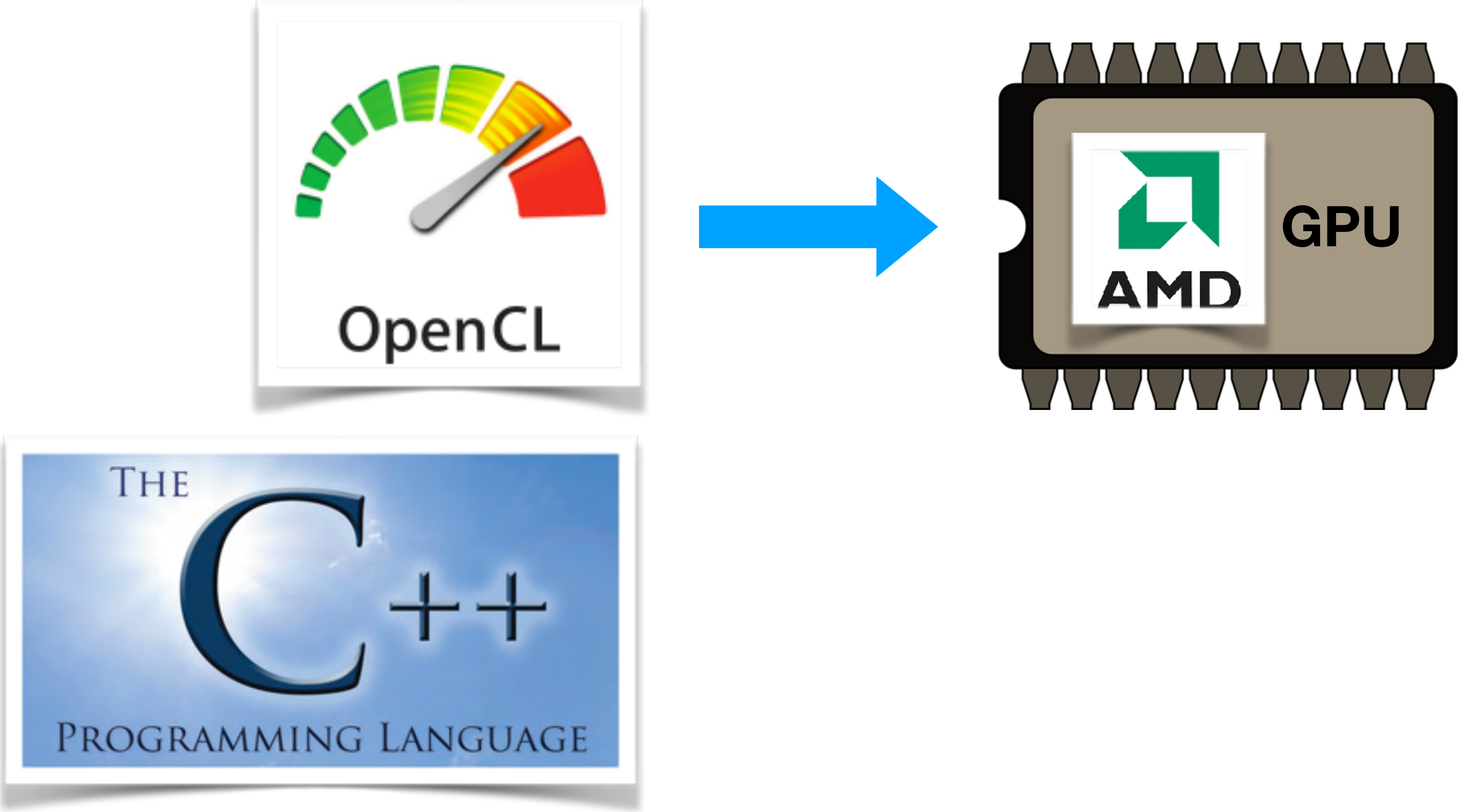
> "Only after the work by John and others in the programming languages community, have programmers gained confidence in using fine-grain inter-thread communication and synchronization on GPUs." *Dr Brad Beckmann, Principal Researcher, AMD*

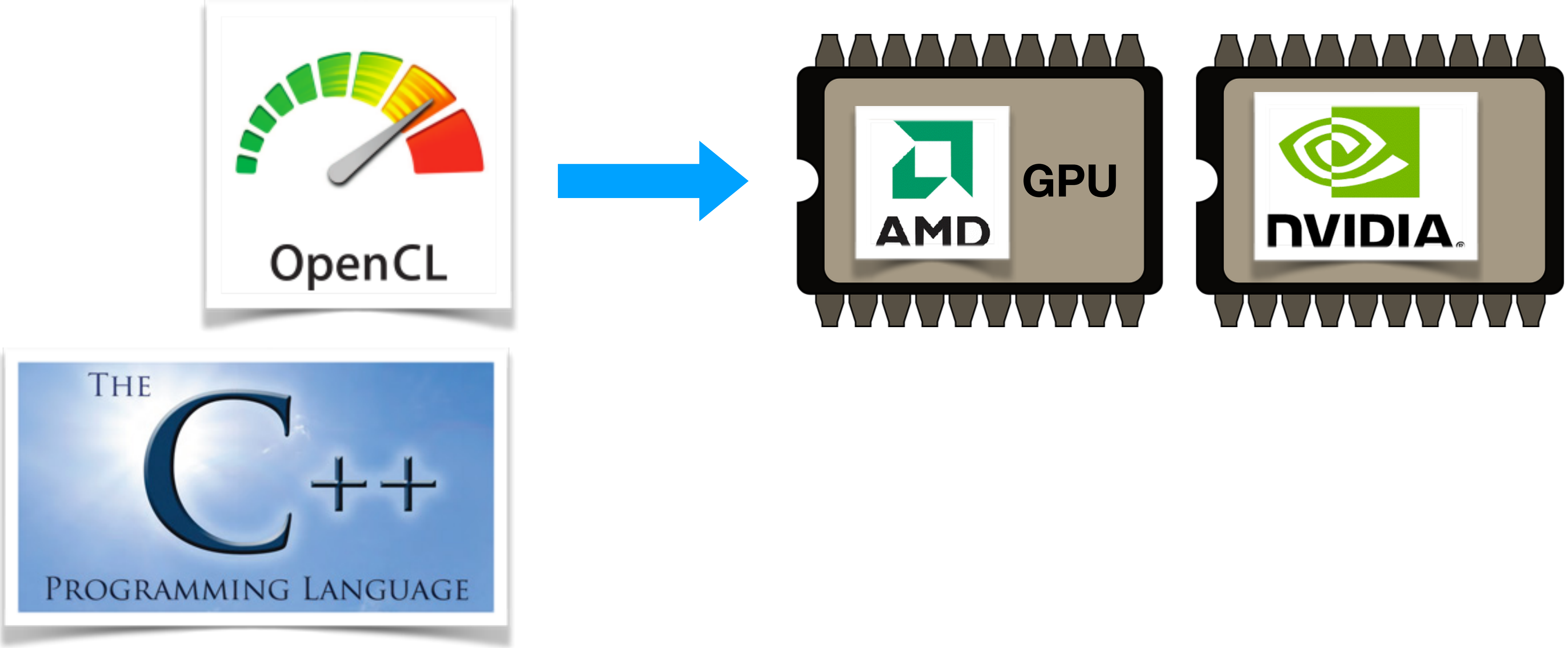


"Only after the work by John and others in the programming languages community, have programmers gained confidence in using fine-grain inter-thread communication and synchronization on GPUs." *Dr Brad Beckmann, Principal Researcher, AMD*

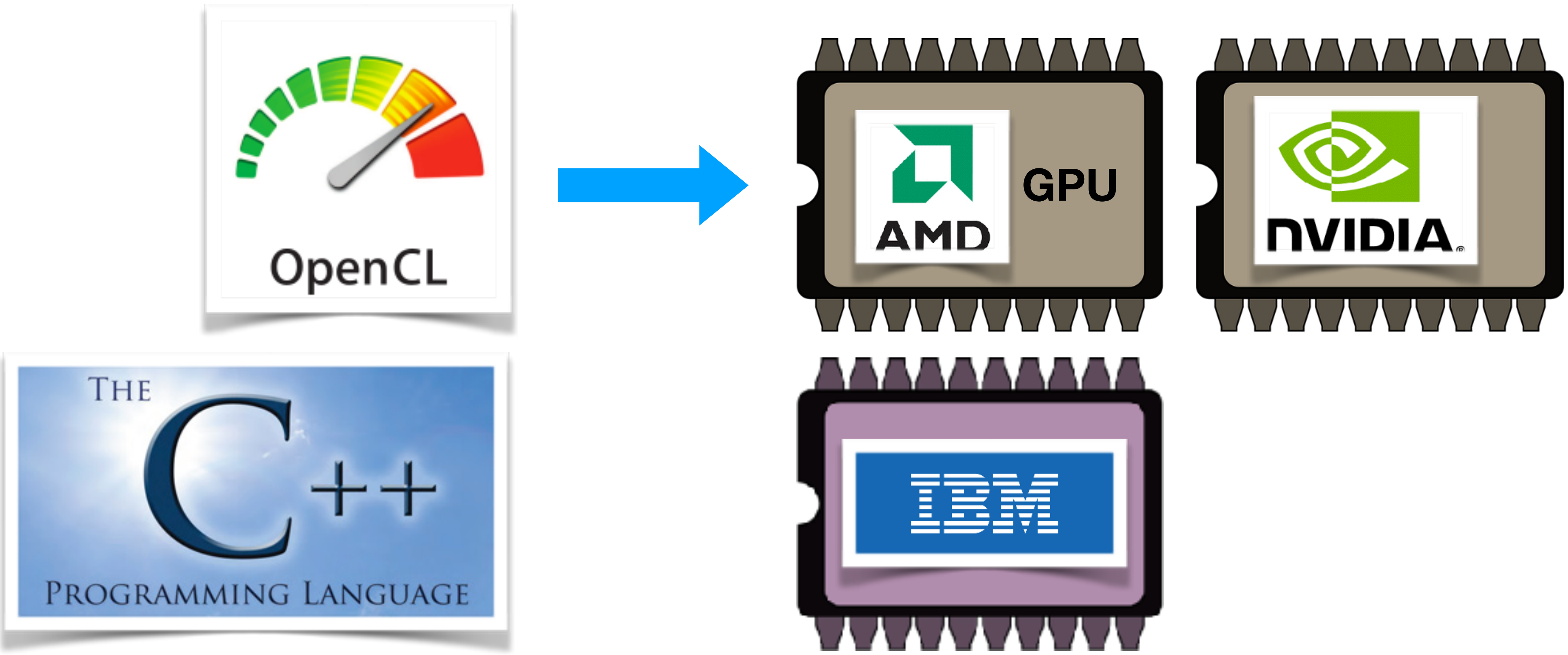
Key publication: ACM **OOPSLA** '15 (with Batty, Beckmann, & Donaldson)

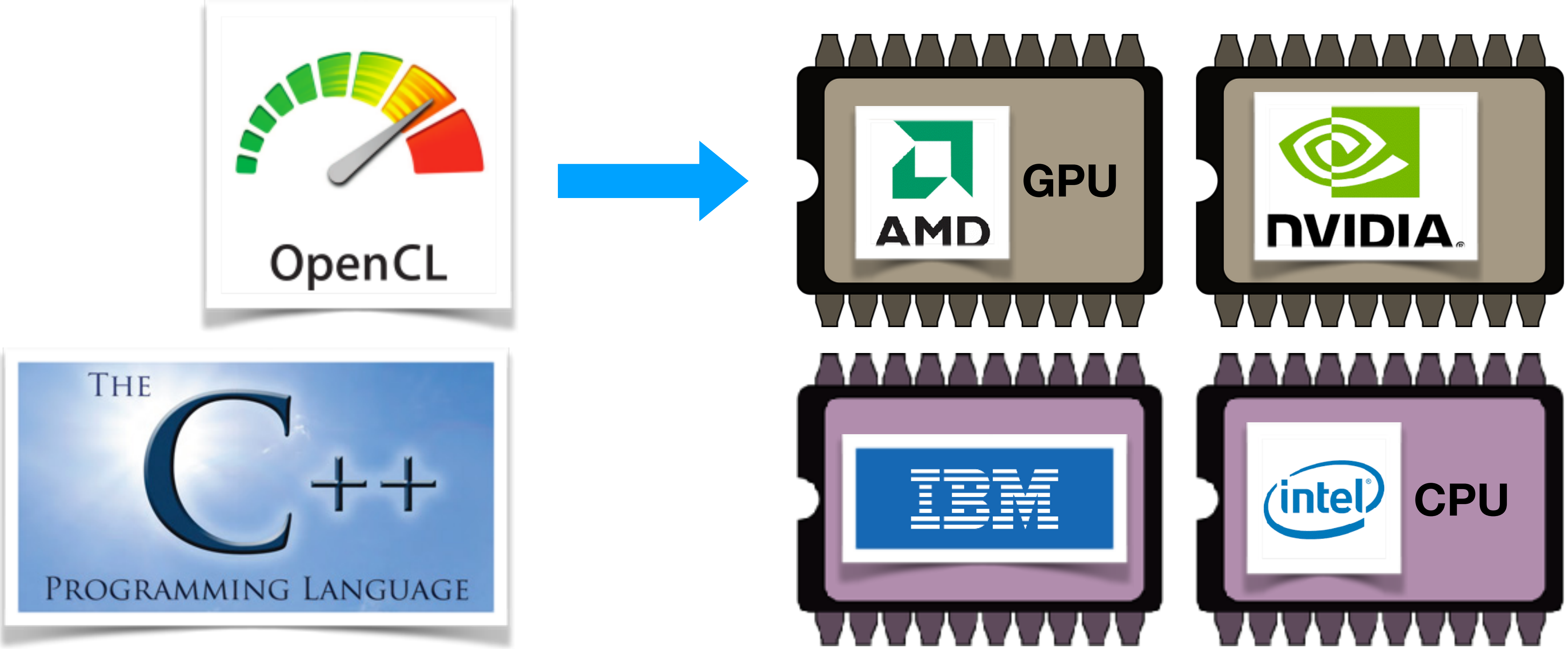

OpenCL
AMD
GPU


OpenCL
THE C++ PROGRAMMING LANGUAGE
AMD
GPU



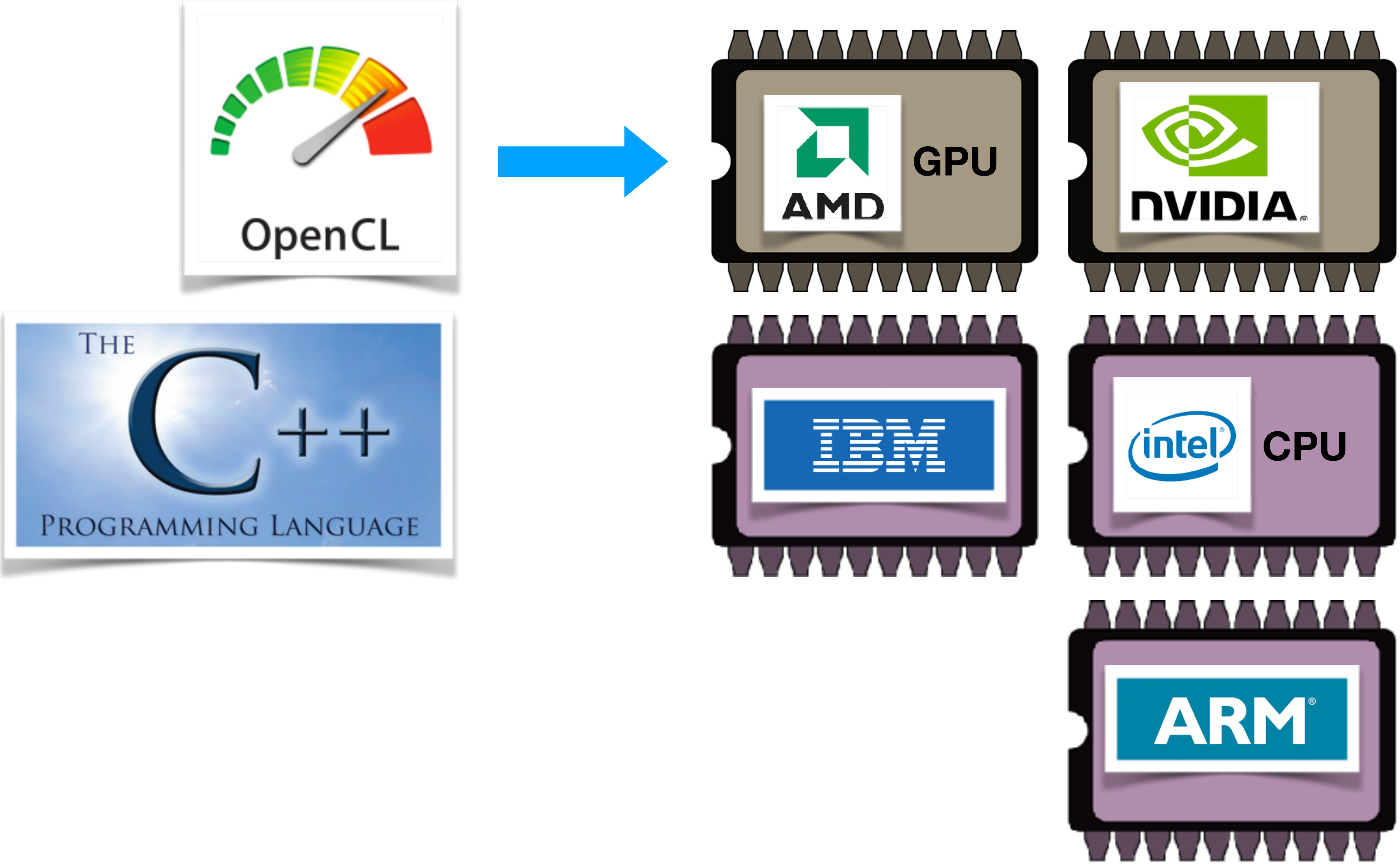
GPU

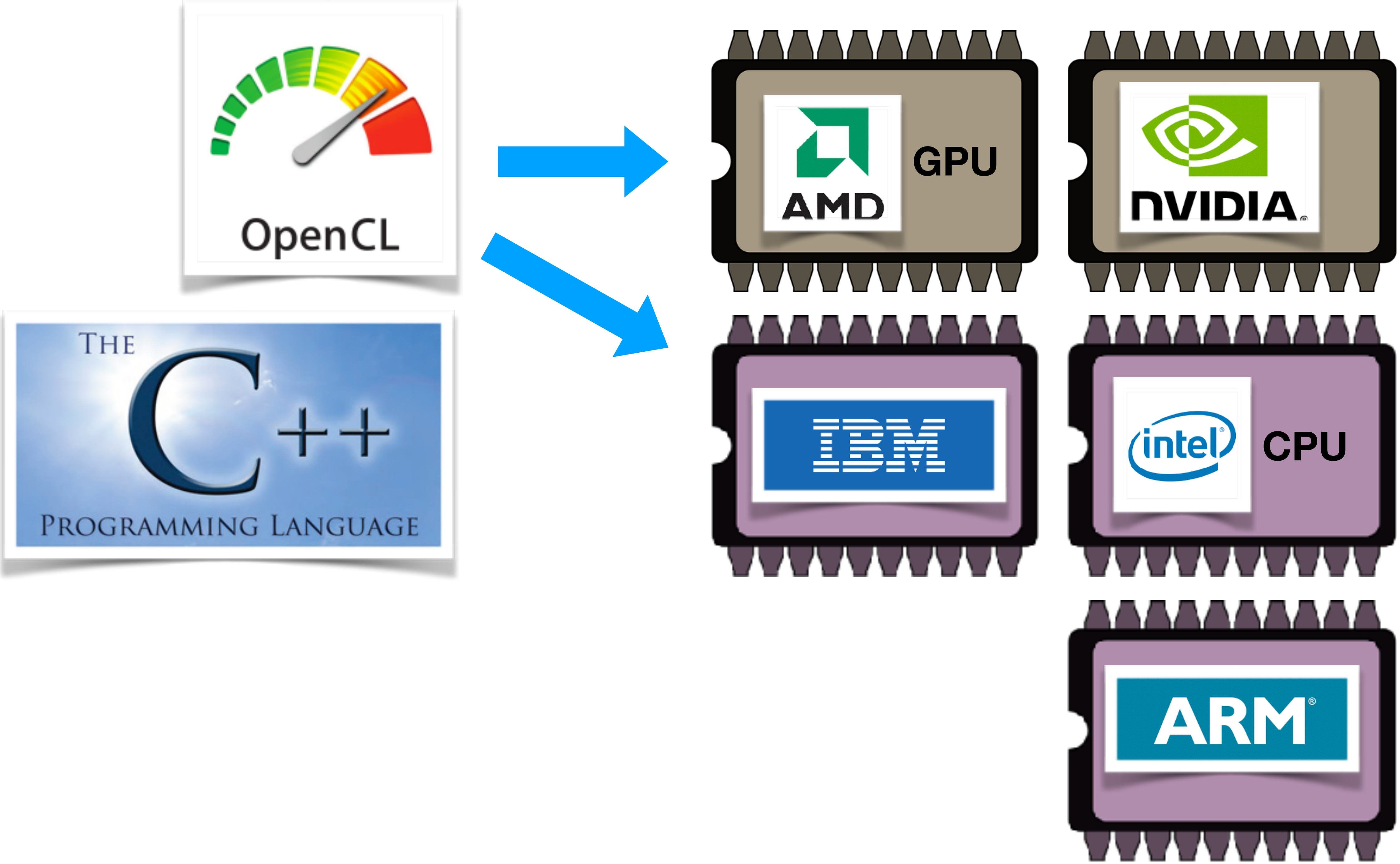

OpenCL
THE C++ PROGRAMMING LANGUAGE
AMD
GPU
NVIDIA
IBM



GPU

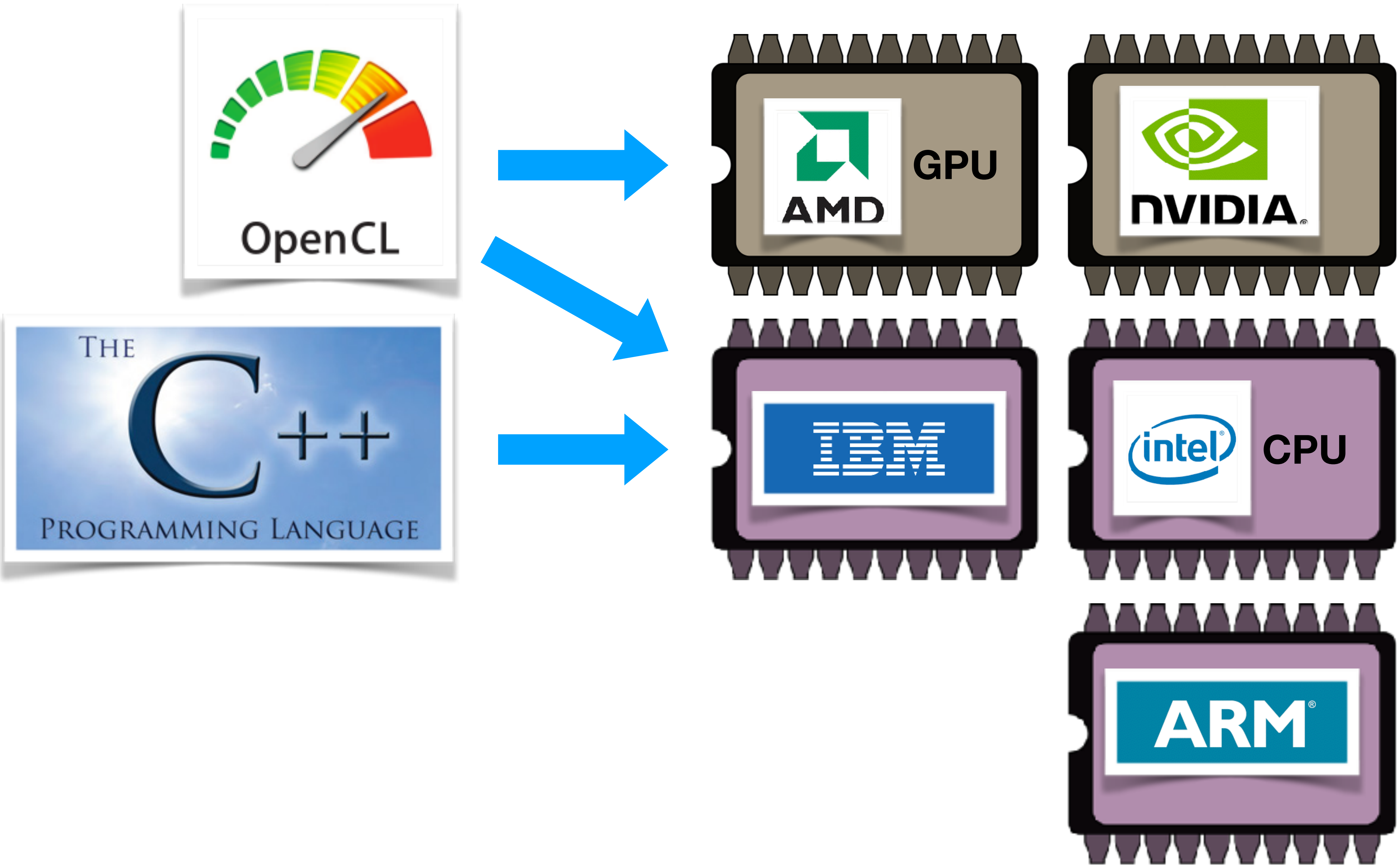
CPU

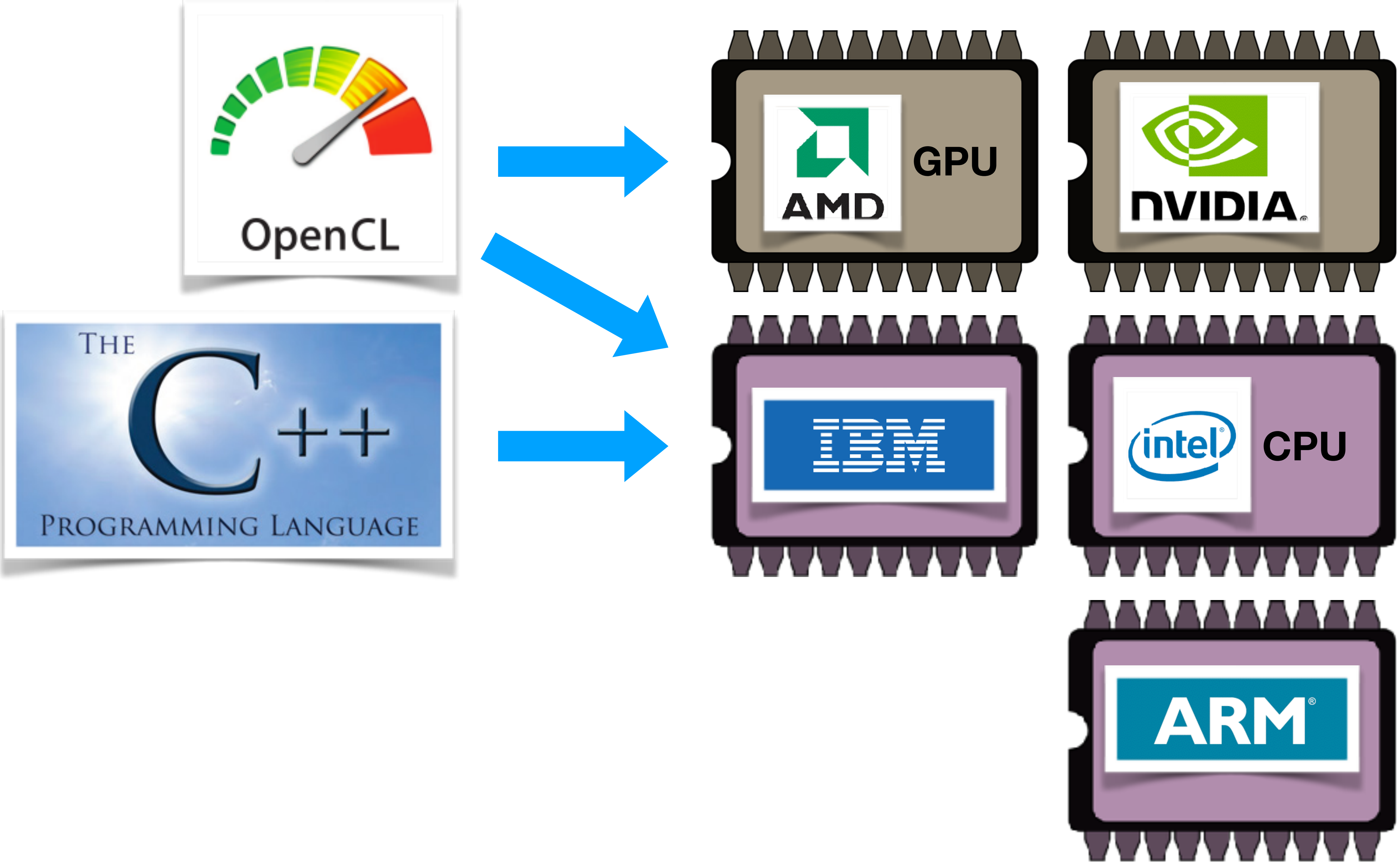

OpenCL
The C++ Programming Language
AMD
GPU
NVIDIA
IBM
intel
CPU
ARM



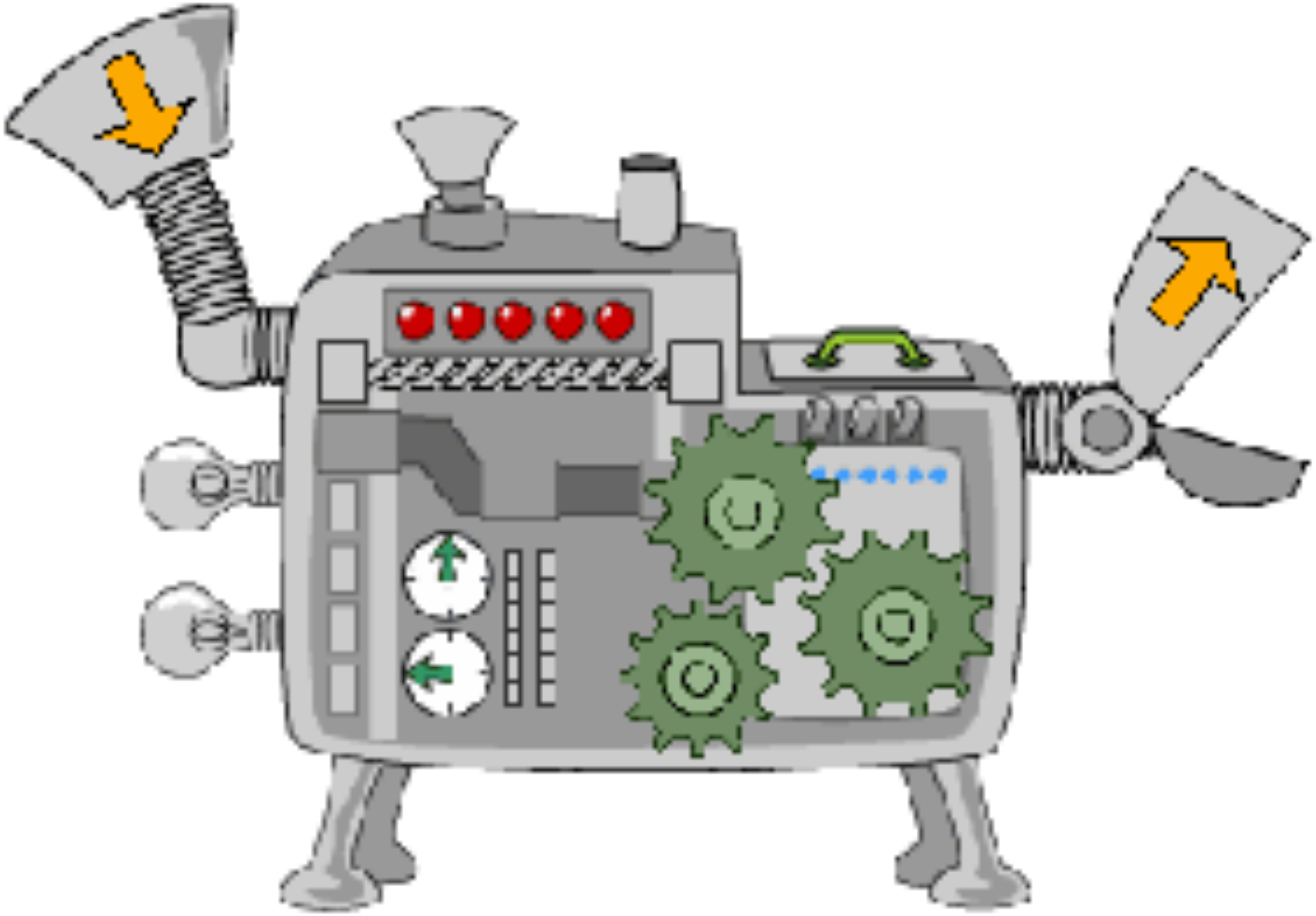
GPU

CPU

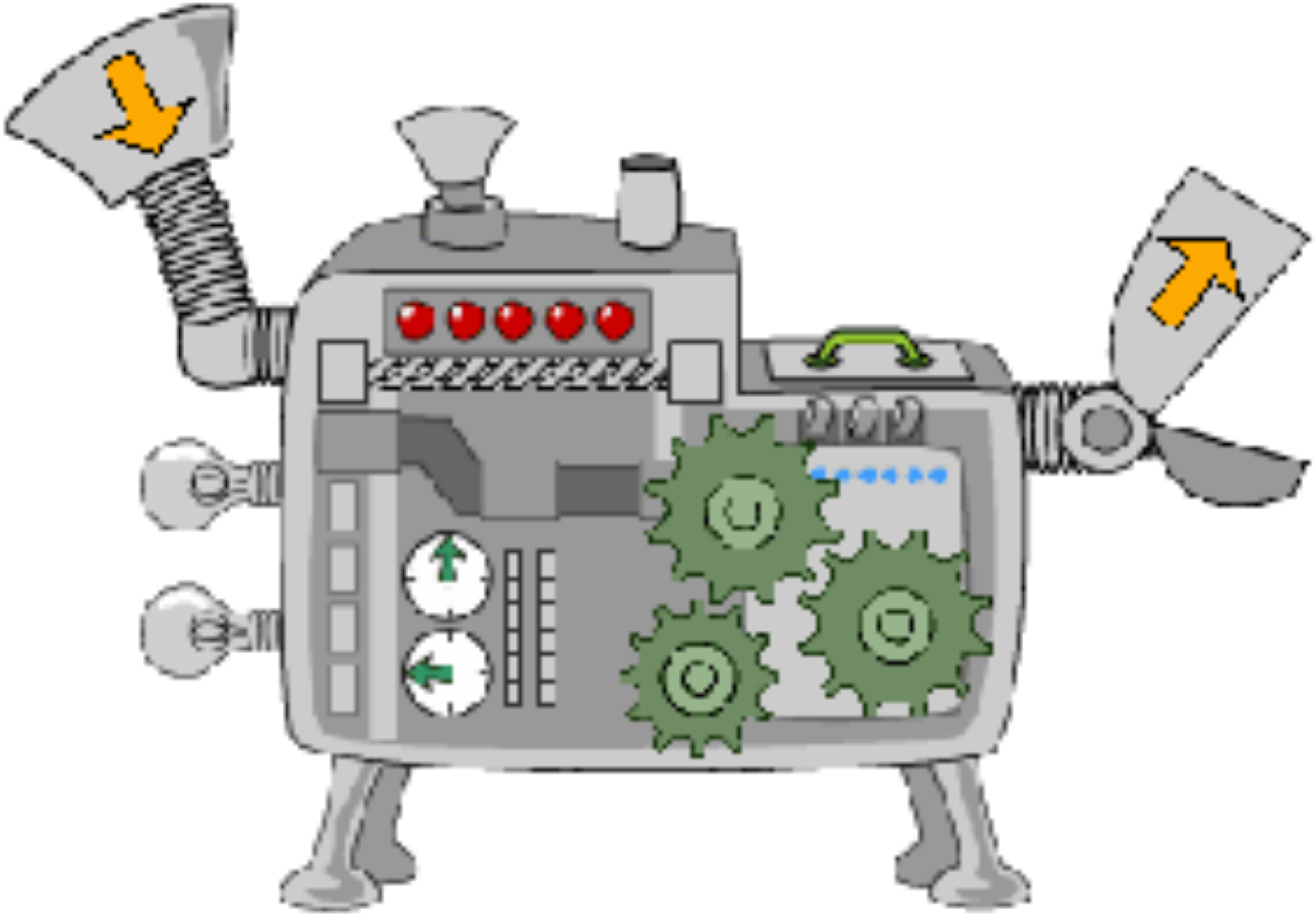

OpenCL
The C++ Programming Language
AMD
GPU
NVIDIA
IBM
intel
CPU
ARM

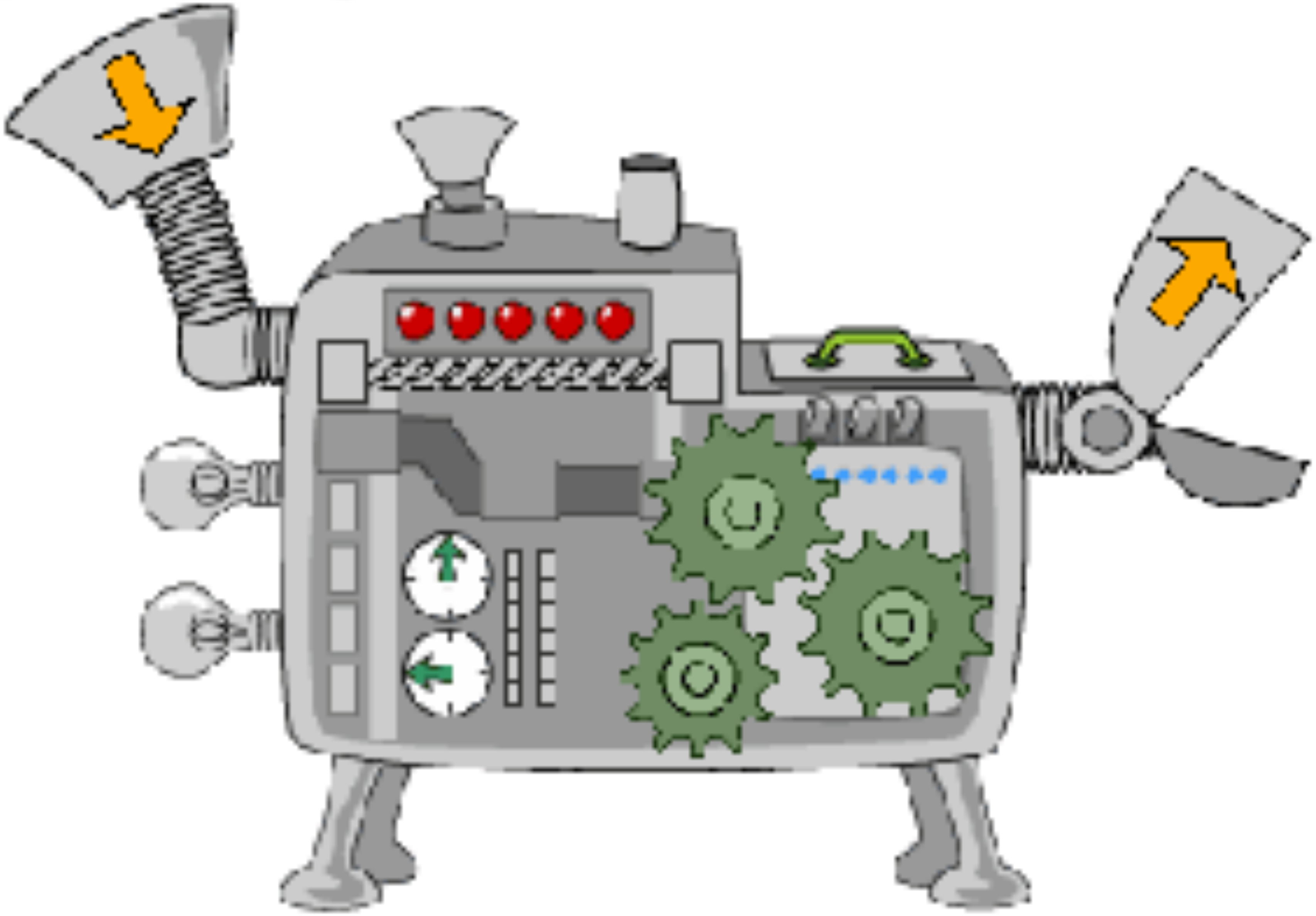


Key publication: ACM **POPL** '17 (with Batty, Sorensen, & Constantinides)



Model of the ARM architecture



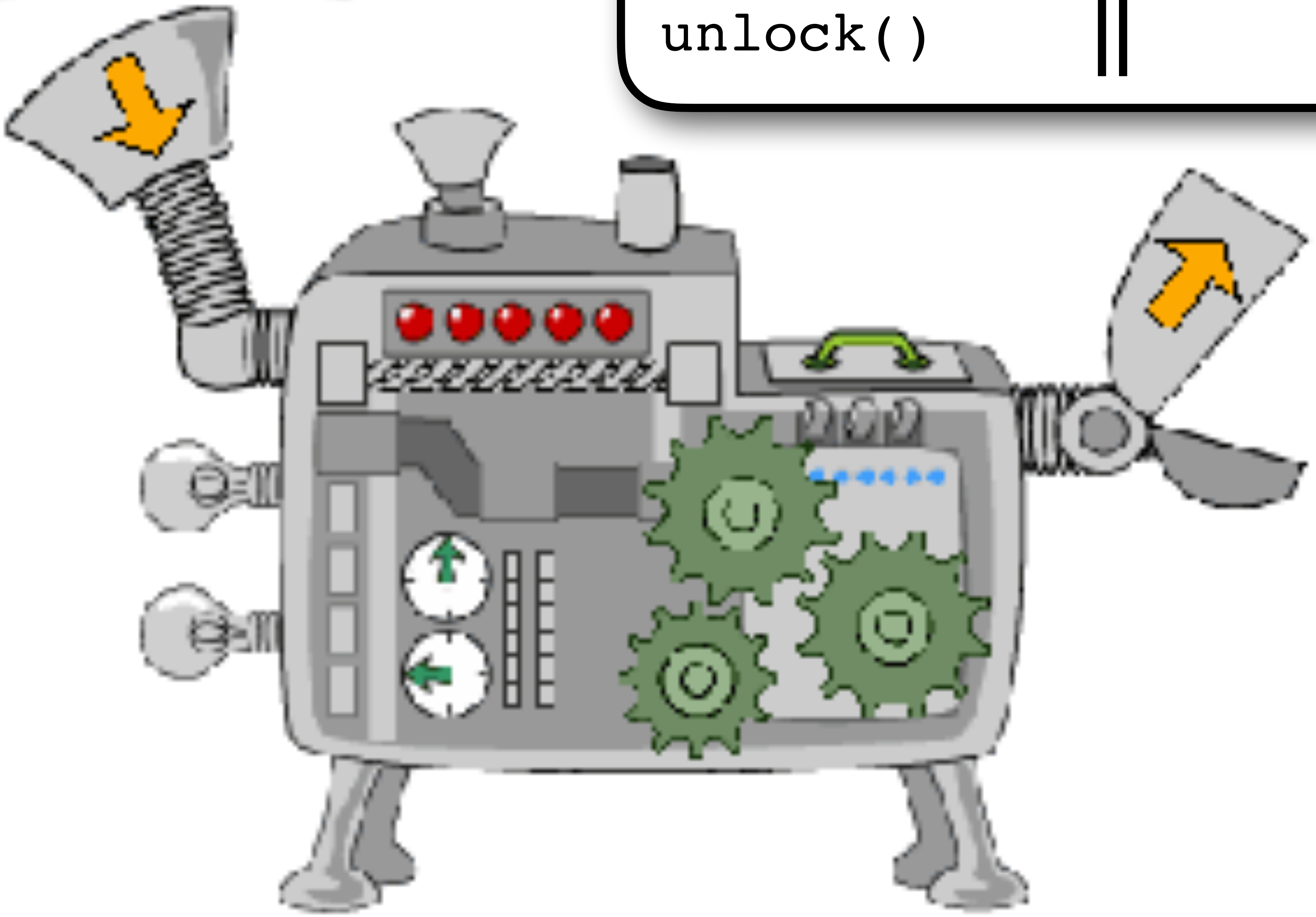


Model of the ARM architecture

Model of the 'transactional lock elision' process

Model of the ARM architecture

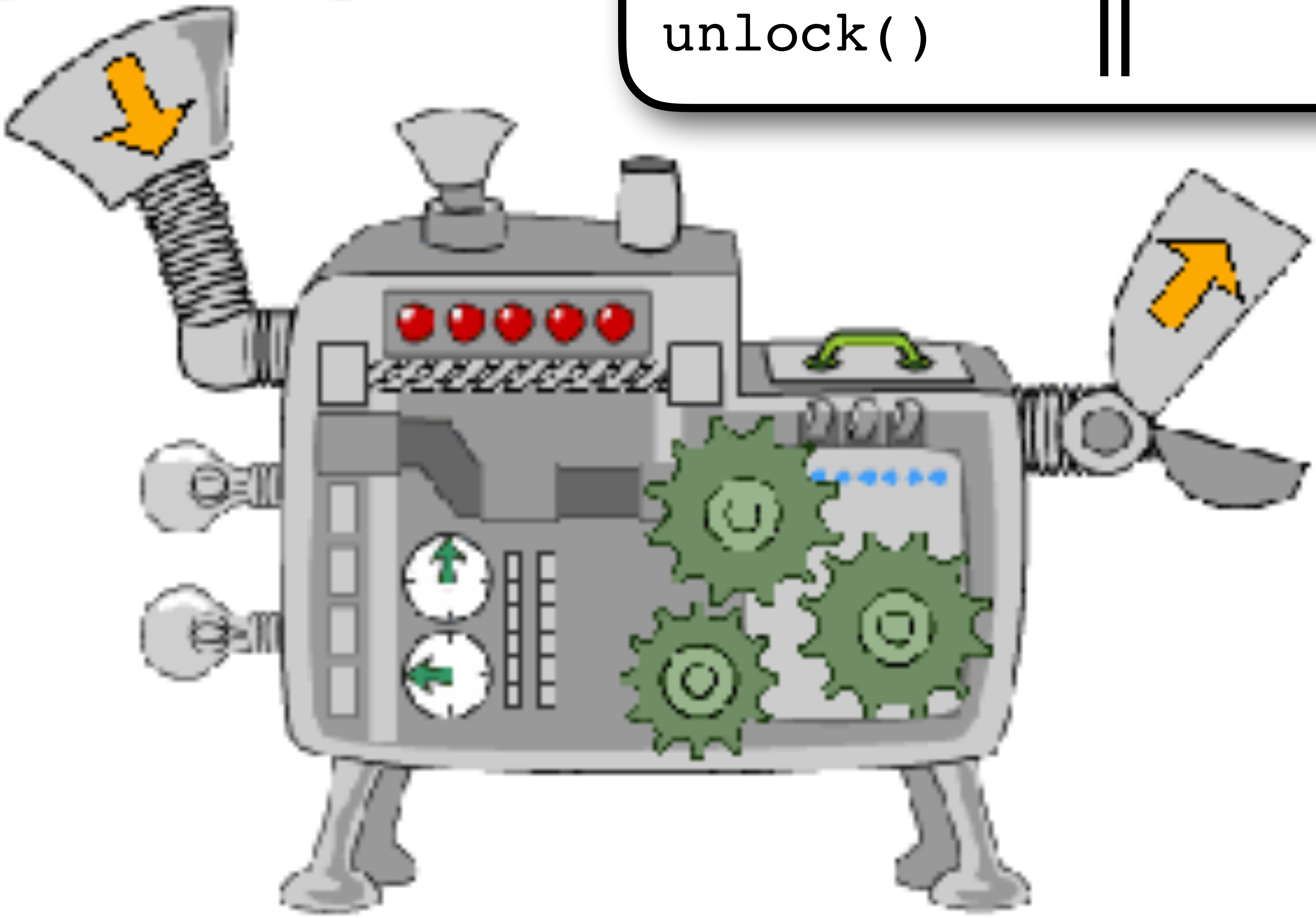
Model of the 'transactional lock elision' process



```
lock()                  ||  lock()
LDR W5,[X0]             ||  MOV W7,#1
ADD W5,W5,#2            ||  STR W7,[X0]
STR W5,[X0]             ||  unlock()
unlock()                ||
```

Model of the ARM architecture

Model of the 'transactional lock elision' process

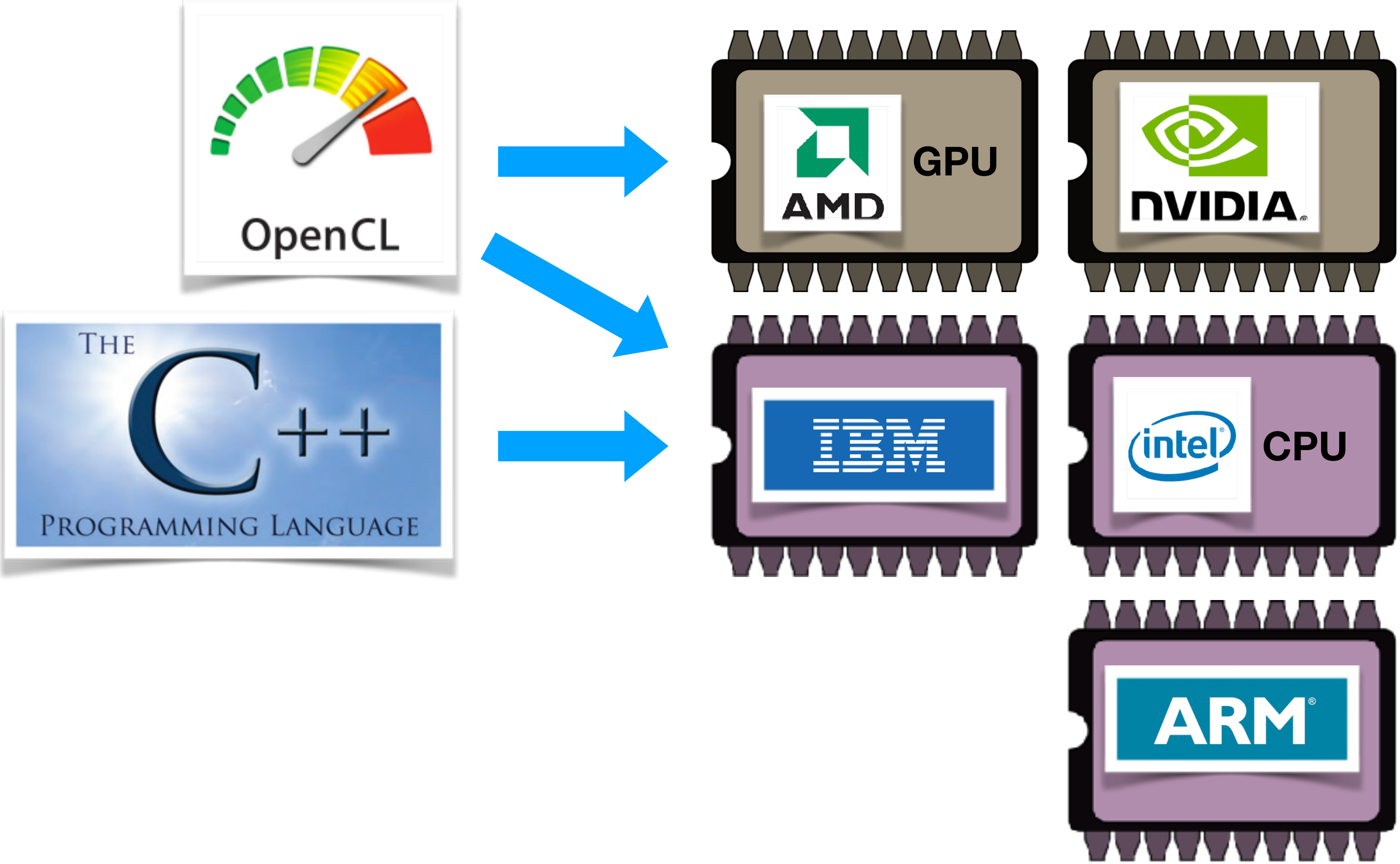


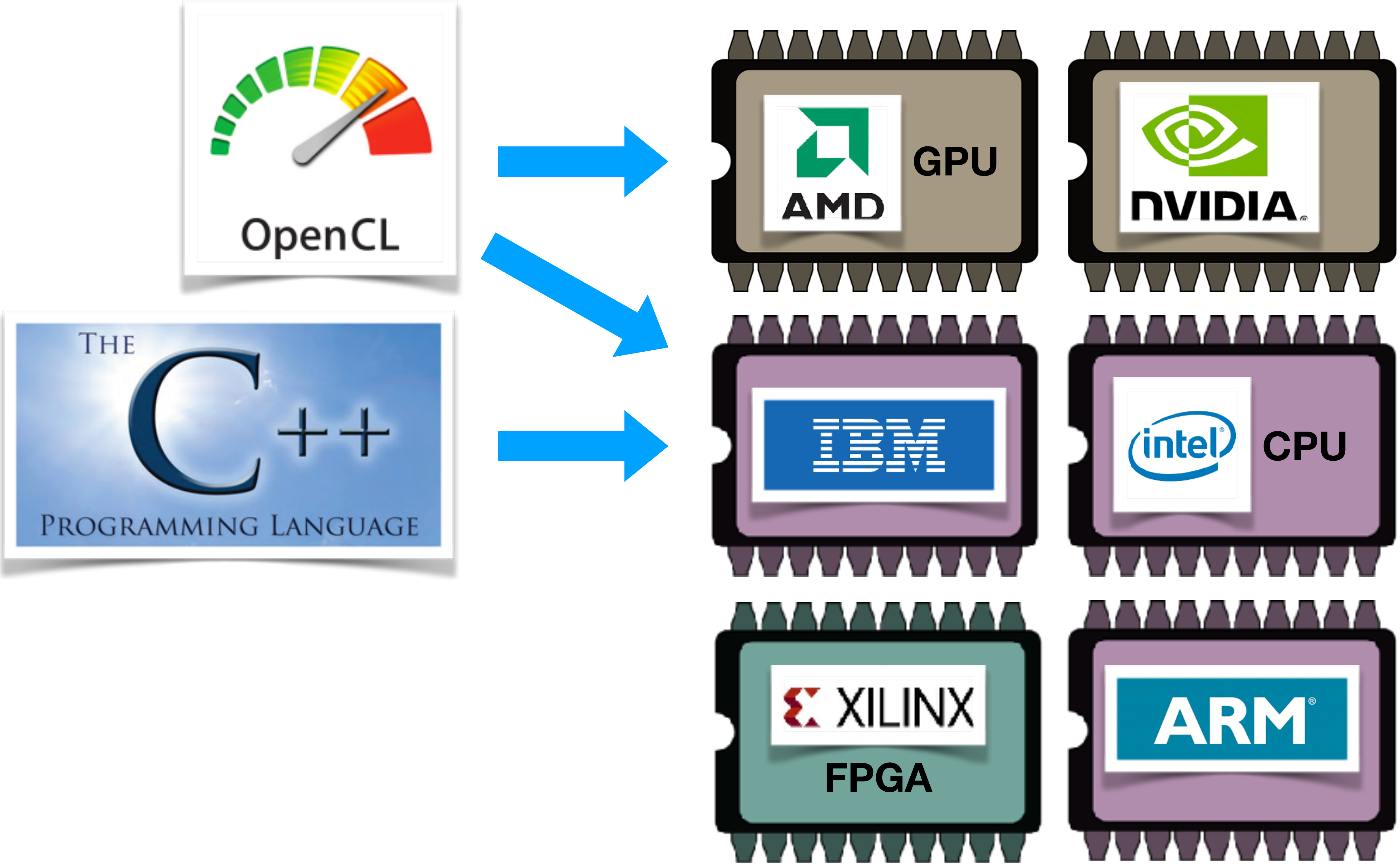
```
lock()          ║ lock()
LDR W5,[X0]     ║ MOV W7,#1
ADD W5,W5,#2    ║ STR W7,[X0]
STR W5,[X0]     ║ unlock()
unlock()        ║
```

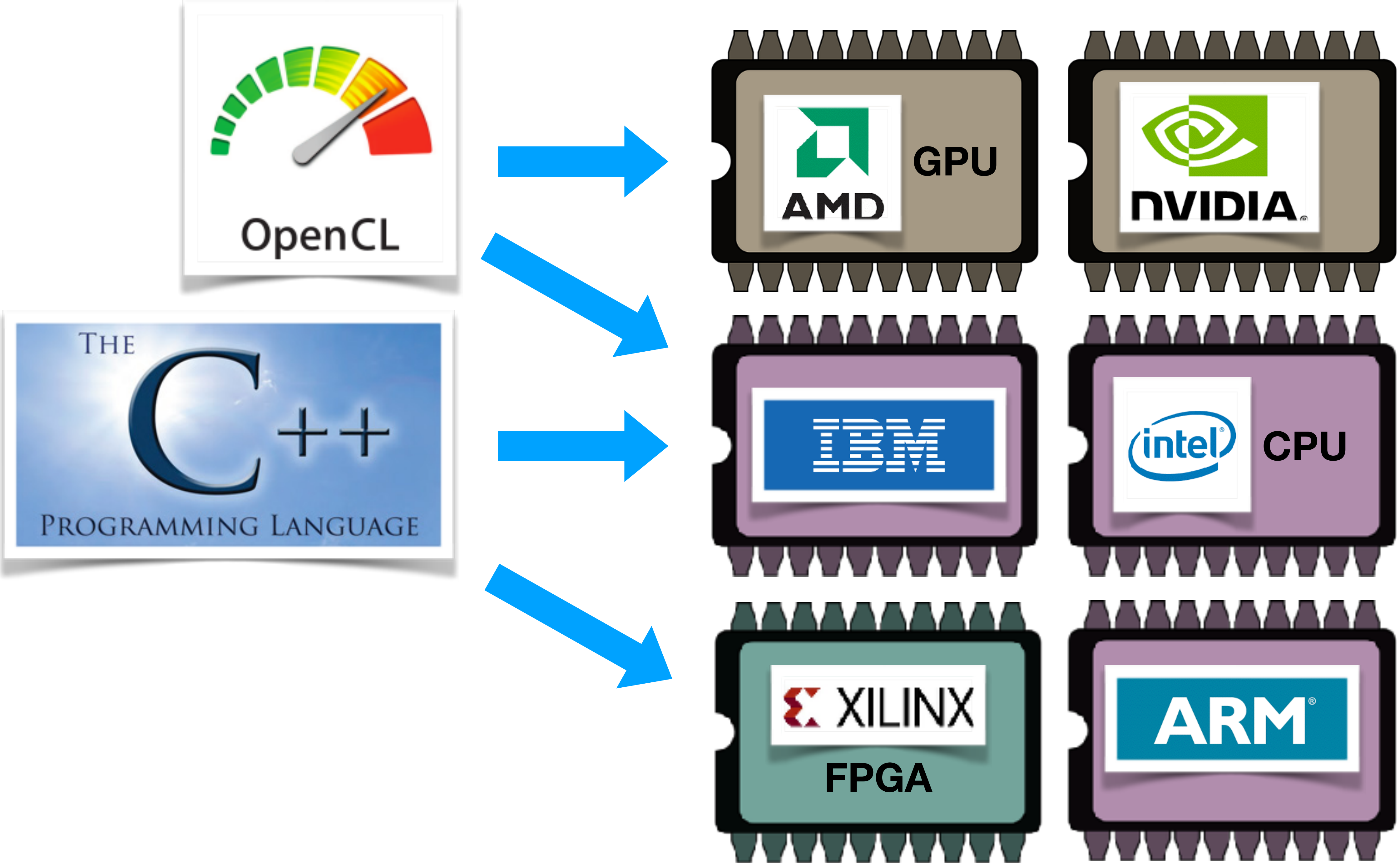
Key publication: ACM **PLDI** '18 (with Chong & Sorensen)

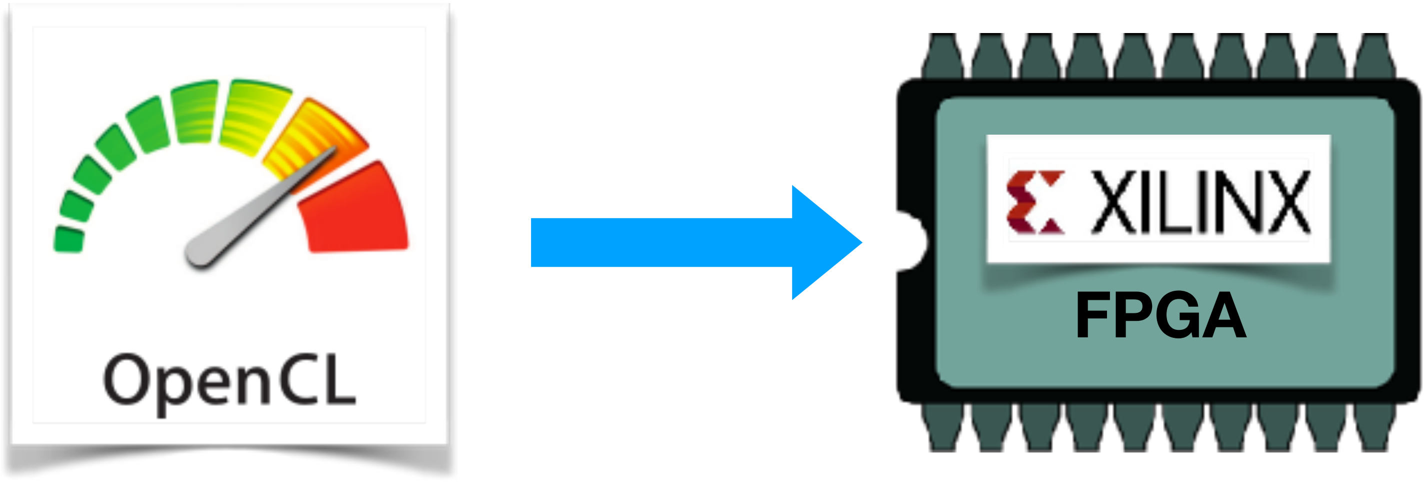
THIS TALK

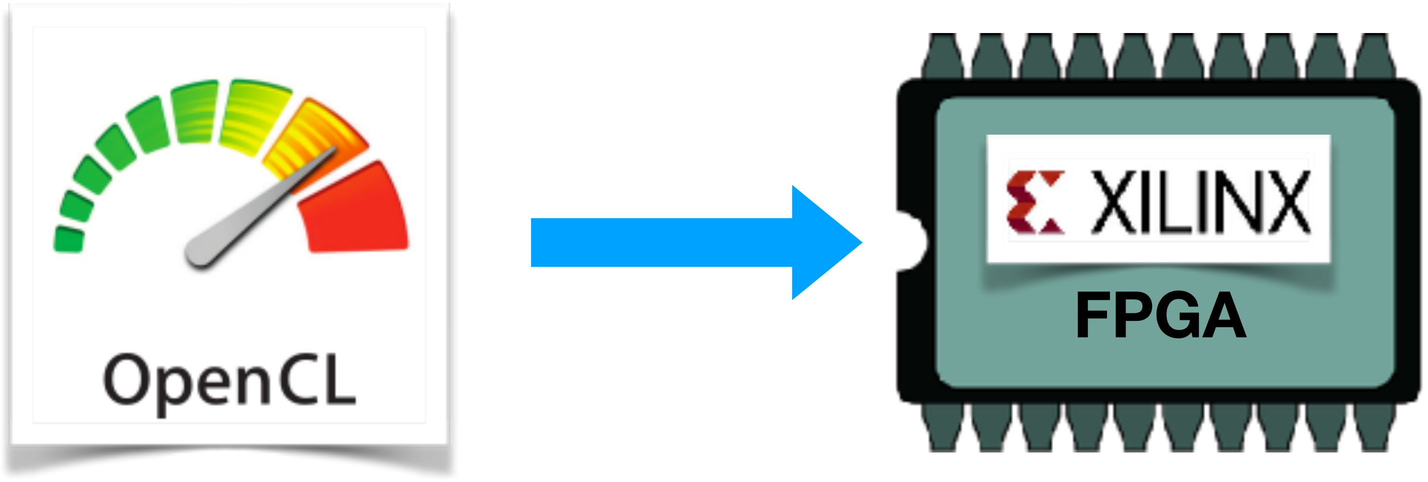
- ~~Weak memory~~
- ~~Formalising the OpenCL memory model~~
- ~~Implementing the OpenCL memory model on GPUs~~
- Implementing the OpenCL memory model on FPGAs


OpenCL
The C++ Programming Language
AMD
GPU
NVIDIA
IBM
intel
CPU
ARM

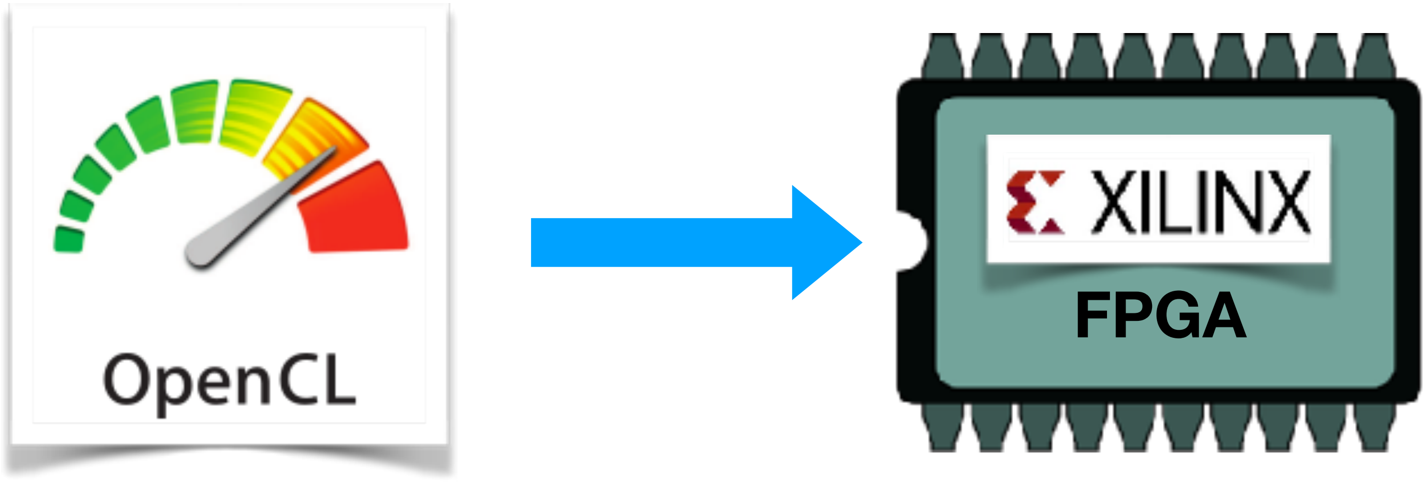

OpenCL
THE C++ PROGRAMMING LANGUAGE
AMD
GPU
NVIDIA
IBM
intel
CPU
XILINX
FPGA
ARM


OpenCL
THE C++ PROGRAMMING LANGUAGE
AMD
GPU
NVIDIA
IBM
intel
CPU
XILINX
FPGA
ARM


OpenCL
XILINX
FPGA

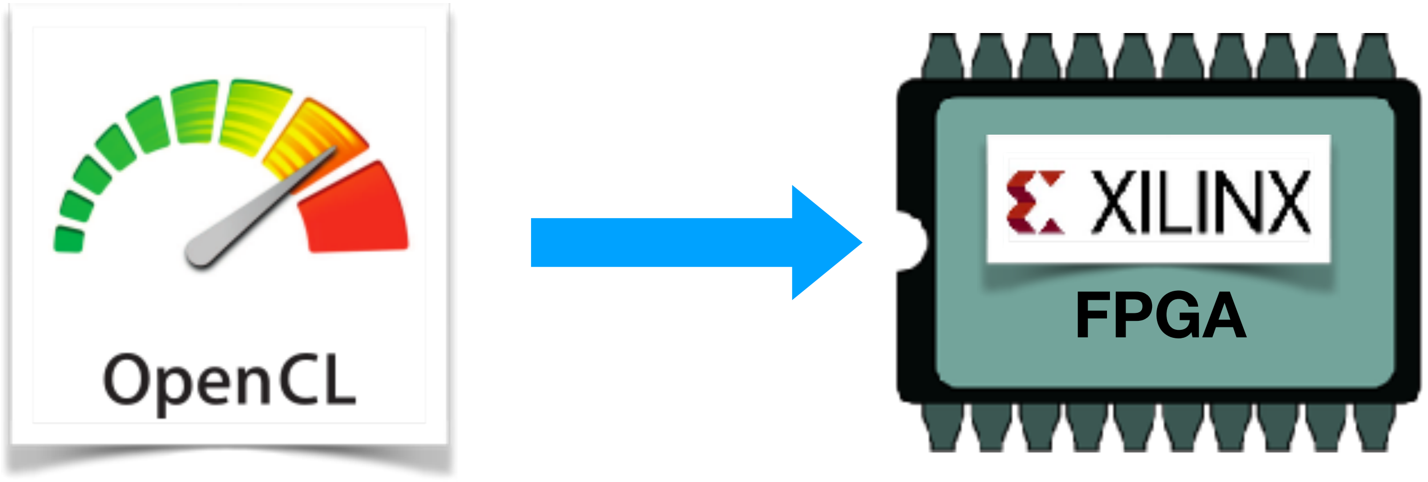


```
r = atomic_load(&y,
  memory_order_acquire);
```



```
r = atomic_load(&y,
  memory_order_acquire);
```

```
**not supported**
```



```
r = atomic_load(&y,
  memory_order_acquire);
```

```
lock();
r = y;
unlock();
```

SCHEDULING ATOMICS

Clock cycle:	1	2	3	4	5	6
`r1 = x`	load `x`					

`r2 = atomic_load(&y,`
`  acquire)`
			load `y`			

`r3 = z`					load `z`	

SCHEDULING ATOMICS

Clock cycle:	1	2	3	4	5	6
`r1 = x`	load x					

`r2 = atomic_load(&y,`
`acquire)`
			load y			

`r3 = z`					load z	

"Too conservative!"

SCHEDULING ATOMICS

Clock cycle:	1	2	3	4	5	6
`r1 = x`	load `x`					
`r2 = atomic_load(&y, acquire)`	load `y`					
`r3 = z`	load `z`					

SCHEDULING ATOMICS

Clock cycle:	1	2	3	4	5	6
`r1 = x`	load `x`					
`r2 = atomic_load(&y, acquire)`	load `y`					
`r3 = z`	load `z`					

"Too aggressive!"

SCHEDULING ATOMICS

Clock cycle:	1	2	3	4	5	6
`r1 = x`	load `x`					
`r2 = atomic_load(&y, acquire)`	load `y`					
`r3 = z`			load `z`			

SCHEDULING ATOMICS

Clock cycle:	1	2	3	4	5	6
`r1 = x`	load `x`					
`r2 = atomic_load(&y, acquire)`	load `y`					
`r3 = z`			load `z`			

"Just right!"

SCHEDULING ATOMICS

Clock cycle:	1	2	3	4	5	6
`r1 = x`	load `x`					
`r2 = atomic_load(&y, acquire)`	load `y`					
`r3 = z`	load `z`					

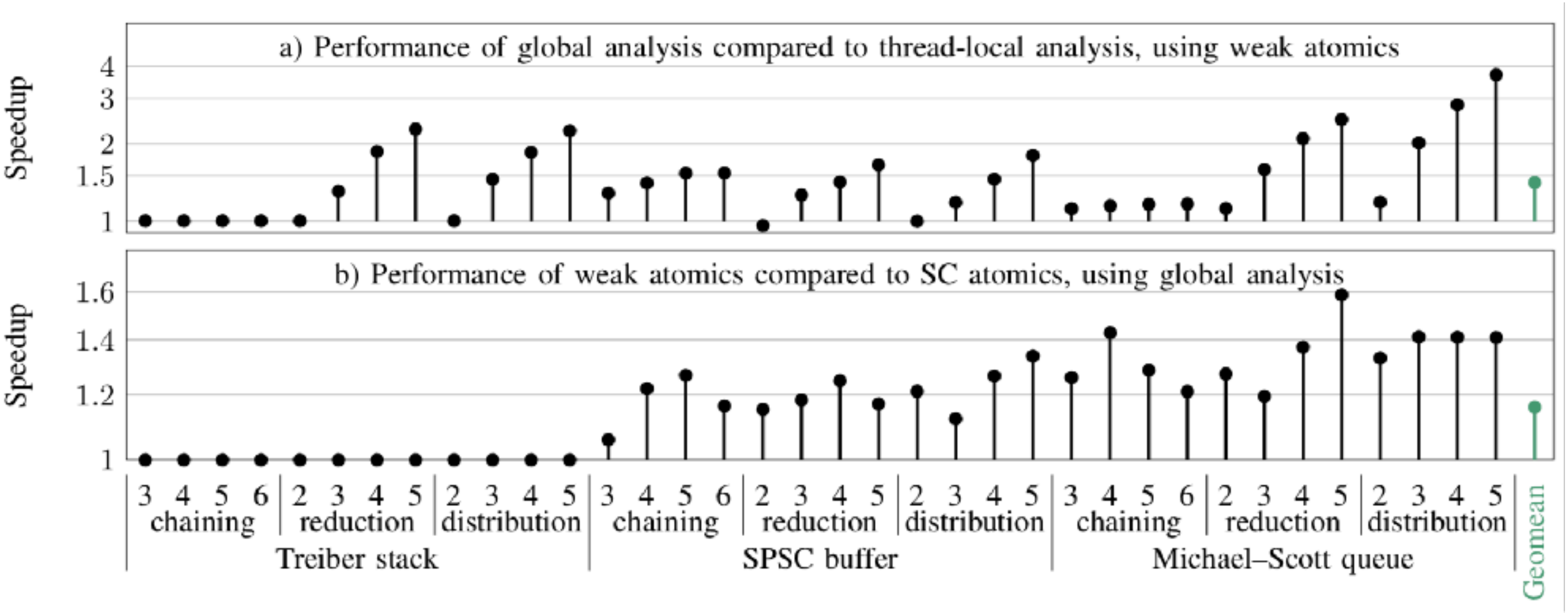
"Too aggressive!"

SCHEDULING ATOMICS

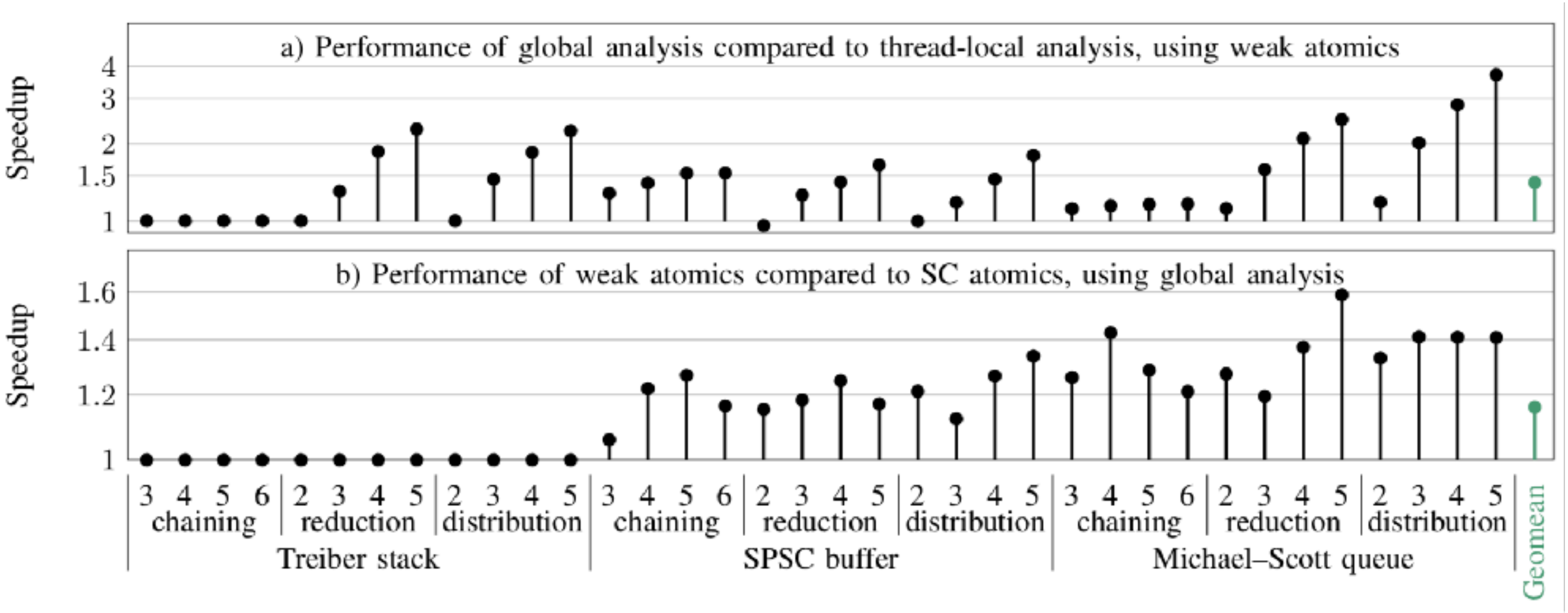
Clock cycle:	1	2	3	4	5	6
`r1 = x`	load x					
`r2 = atomic_load(&y, acquire)`	load y					
`r3 = z`	load z					

"Too aggressive!" ... or is it?

SCHEDULING ATOMICS

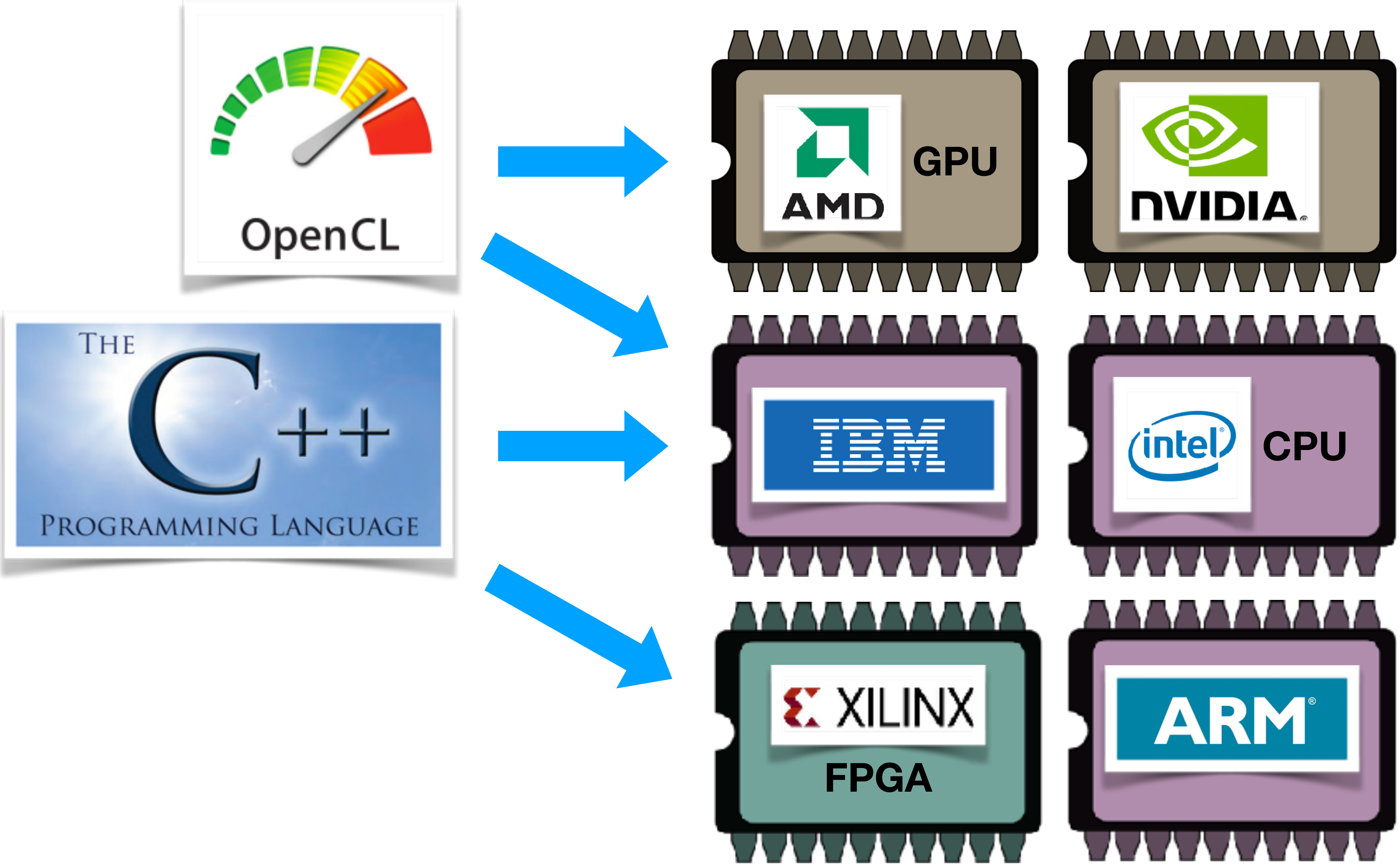


SCHEDULING ATOMICS

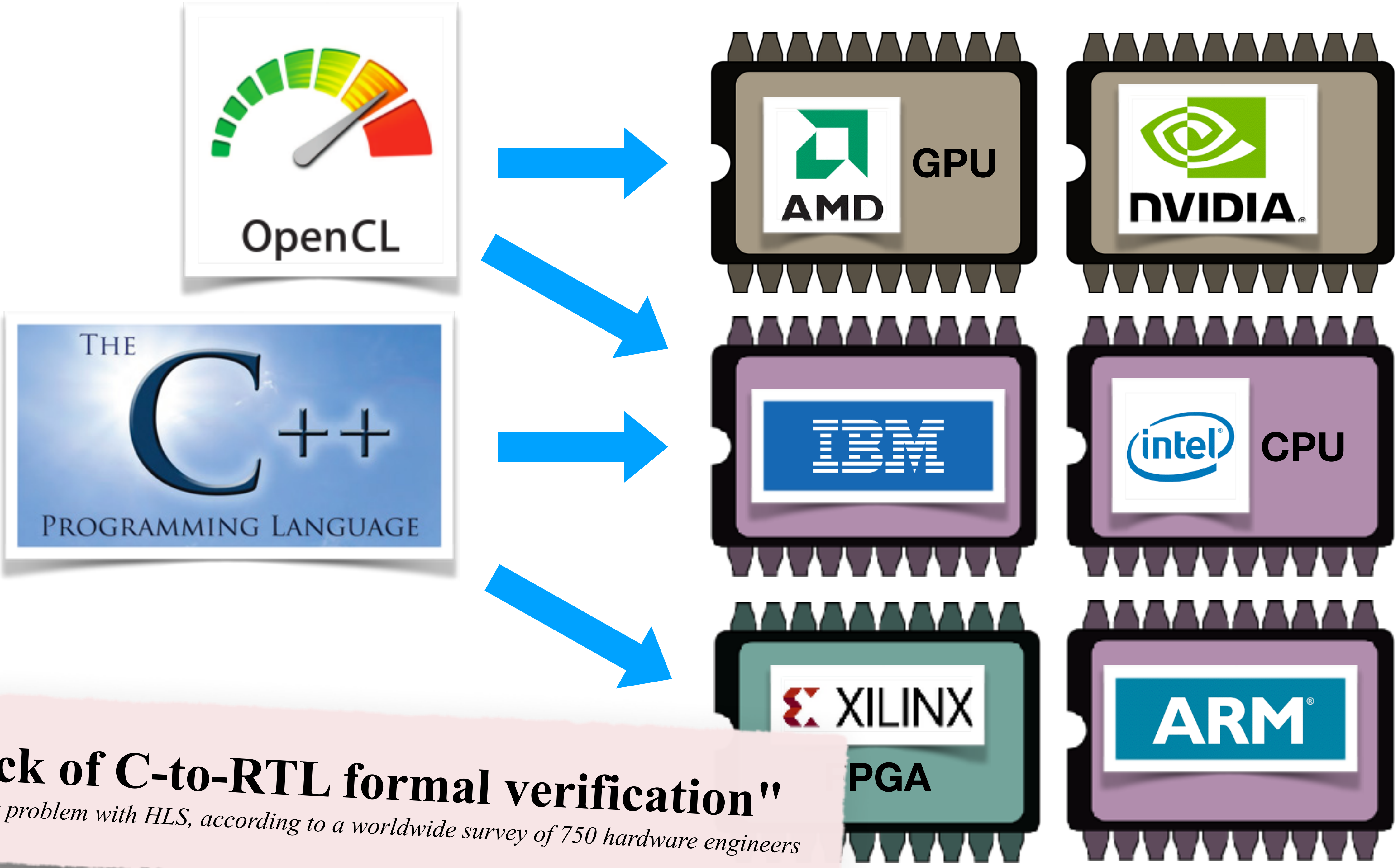


Key publications: ACM **FPGA** '17, IEEE **Trans. on Computers** '17, IEEE **FCCM** '18 (with Ramanathan, Fleming & Constantinides)

THE FUTURE?


OpenCL
THE C++ PROGRAMMING LANGUAGE
AMD
GPU
NVIDIA
IBM
intel
CPU
XILINX
FPGA
ARM

THE FUTURE?



"Lack of C-to-RTL formal verification"

The biggest problem with HLS, according to a worldwide survey of 750 hardware engineers