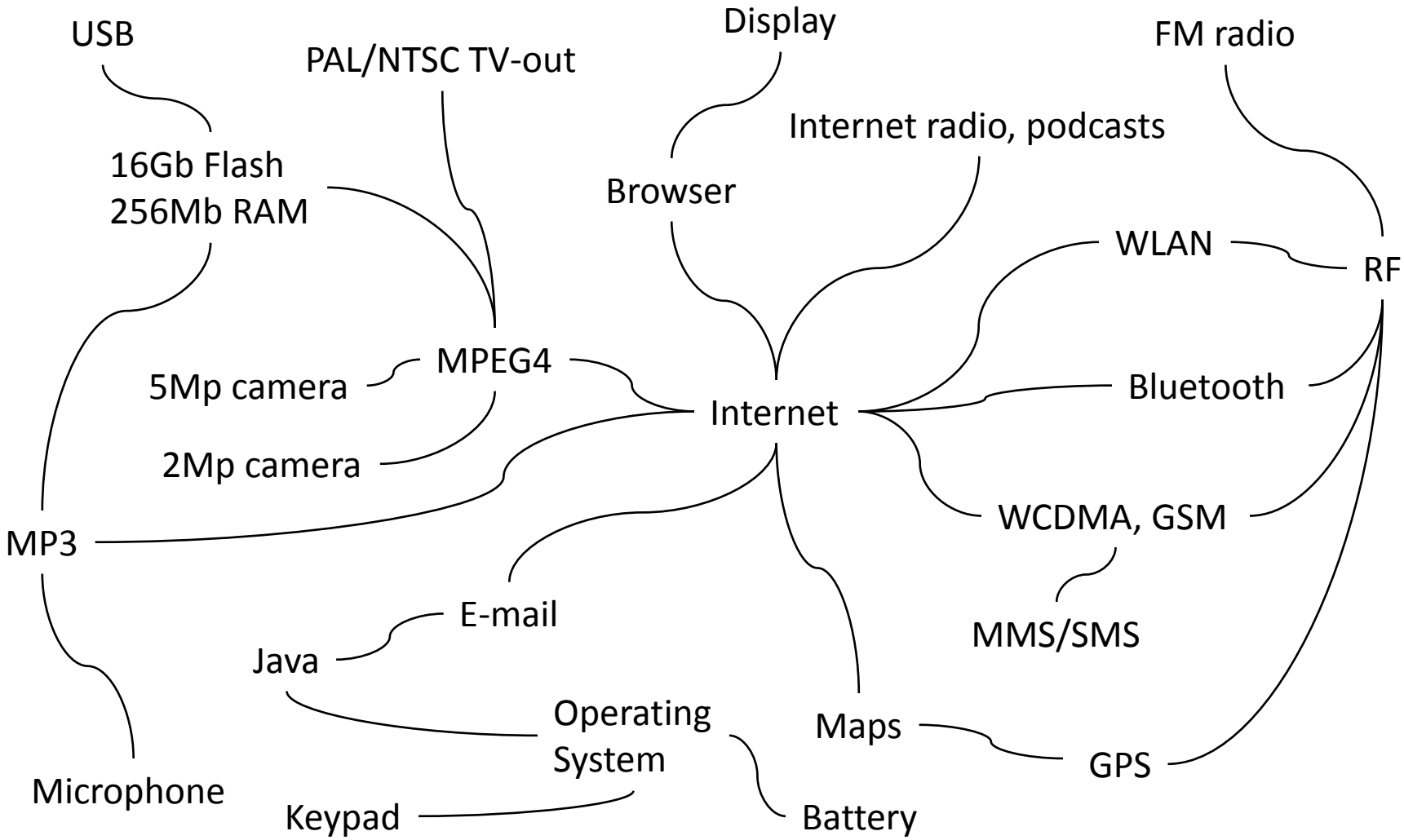
Components and abstraction levels for system-on-chip

Giovanni Funchal
STMicroelectronics
Verimag

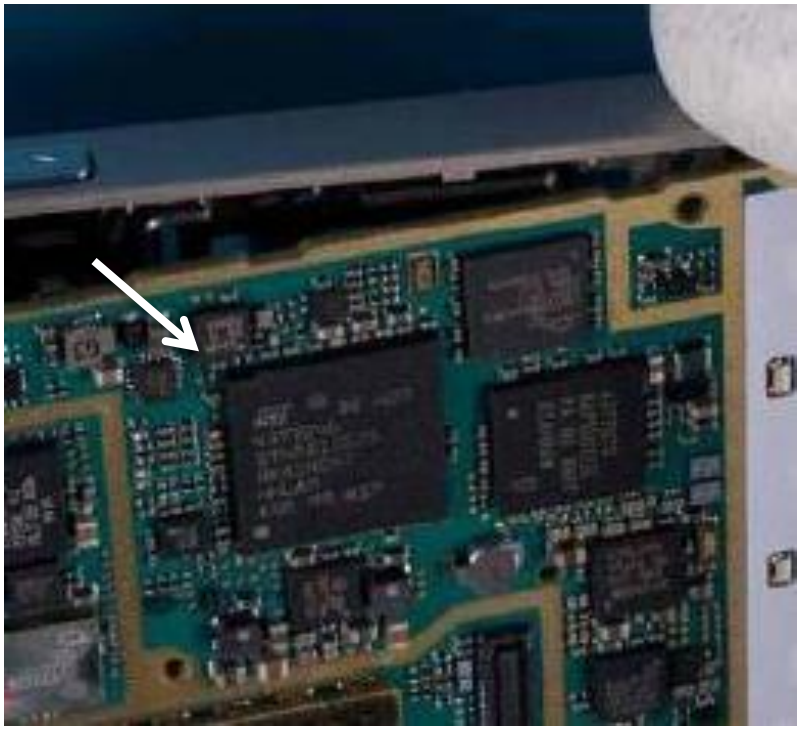
Synchron'2008

Imagine a system…



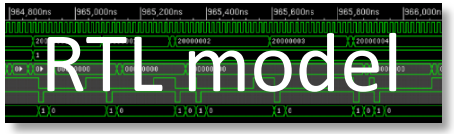
…-on-chip

Nokia N96

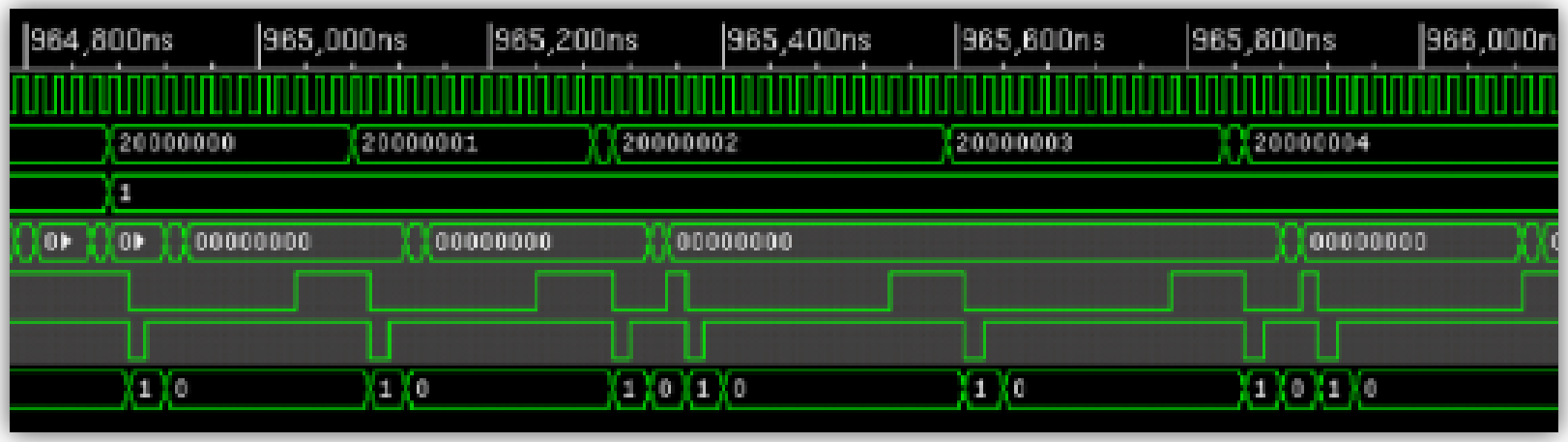


STMicroelectronics
Nomadik STn8815

Design flow



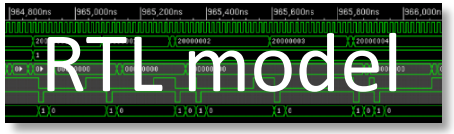
Register transfer level



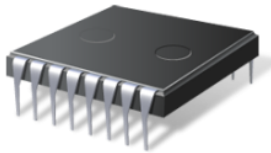
Precise description of hardware

Synchronous, deterministic

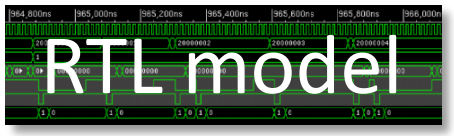
Design flow



→ Fabrication →

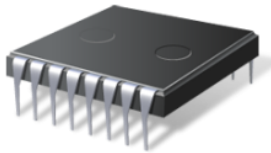


Design flow



Fabrication

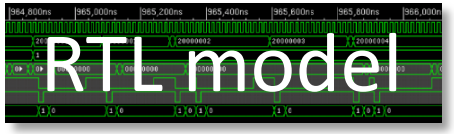
Software



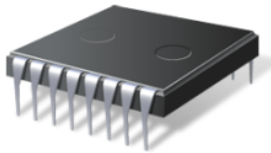
Design flow

Design flow

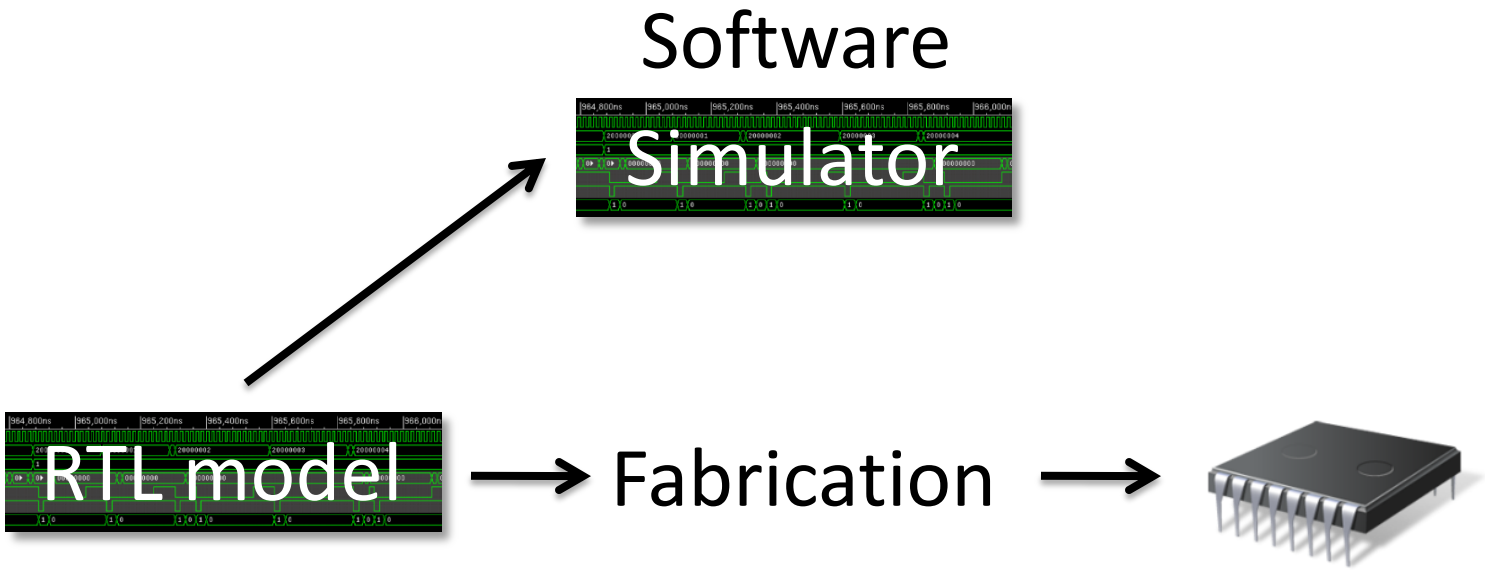
Software



→ Fabrication →



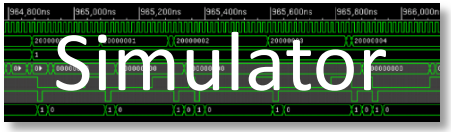
Design flow

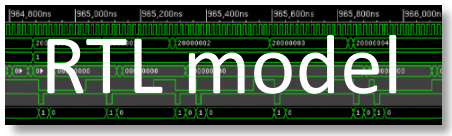


Design flow

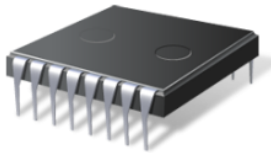
Too slow!

Software



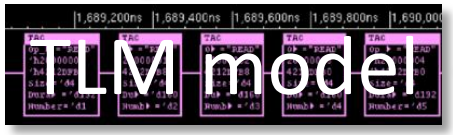


Fabrication



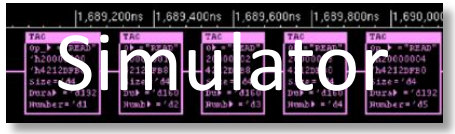
Design flow

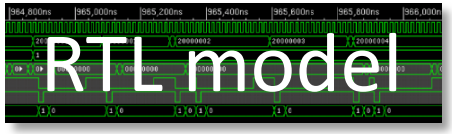
Simpler, faster



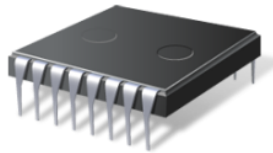
→

Software





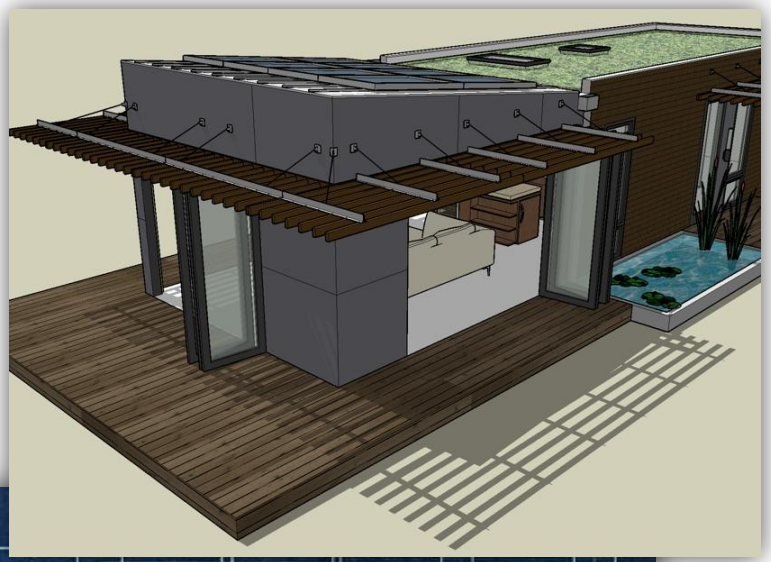
→ Fabrication →



Abstraction levels

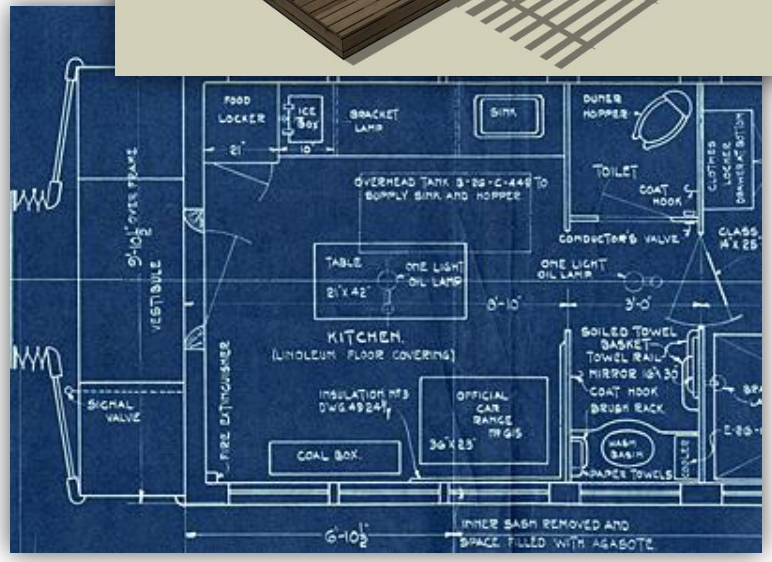
TLM models

- Early (sw dev)
- Less precise
- Fast

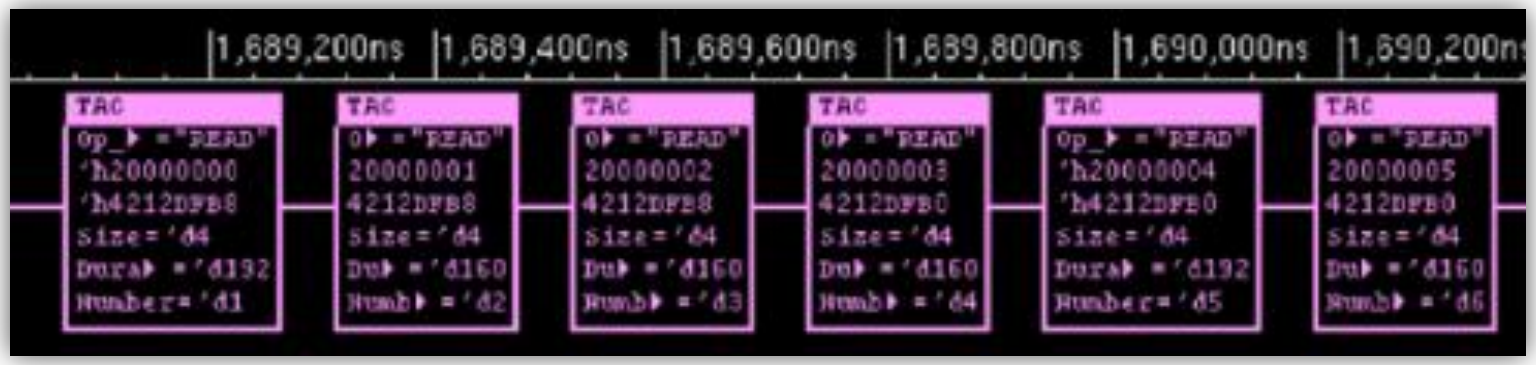


RTL models

- Late (hw fab)
- Precise
- Slow



Transaction level modeling



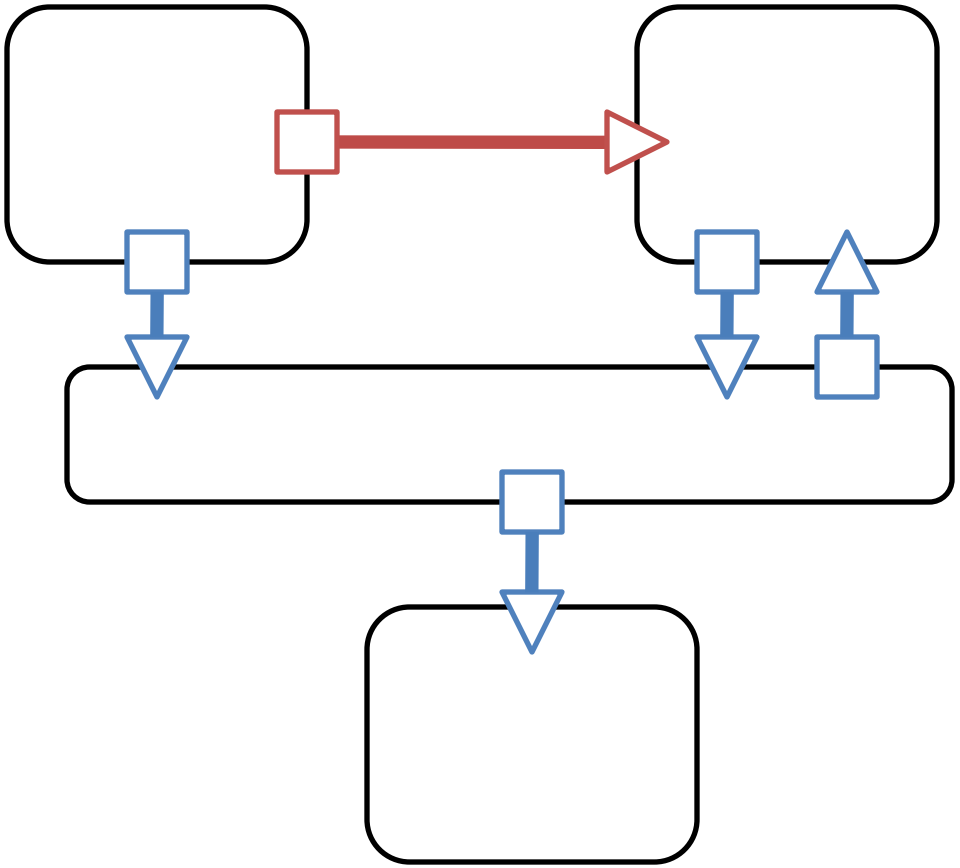
Abstract transfers of data
Asynchronous, non-deterministic

Components and abstraction levels for system-on-chip

Giovanni Funchal
STMicroelectronics
Verimag

Synchron'2008

Components



Component

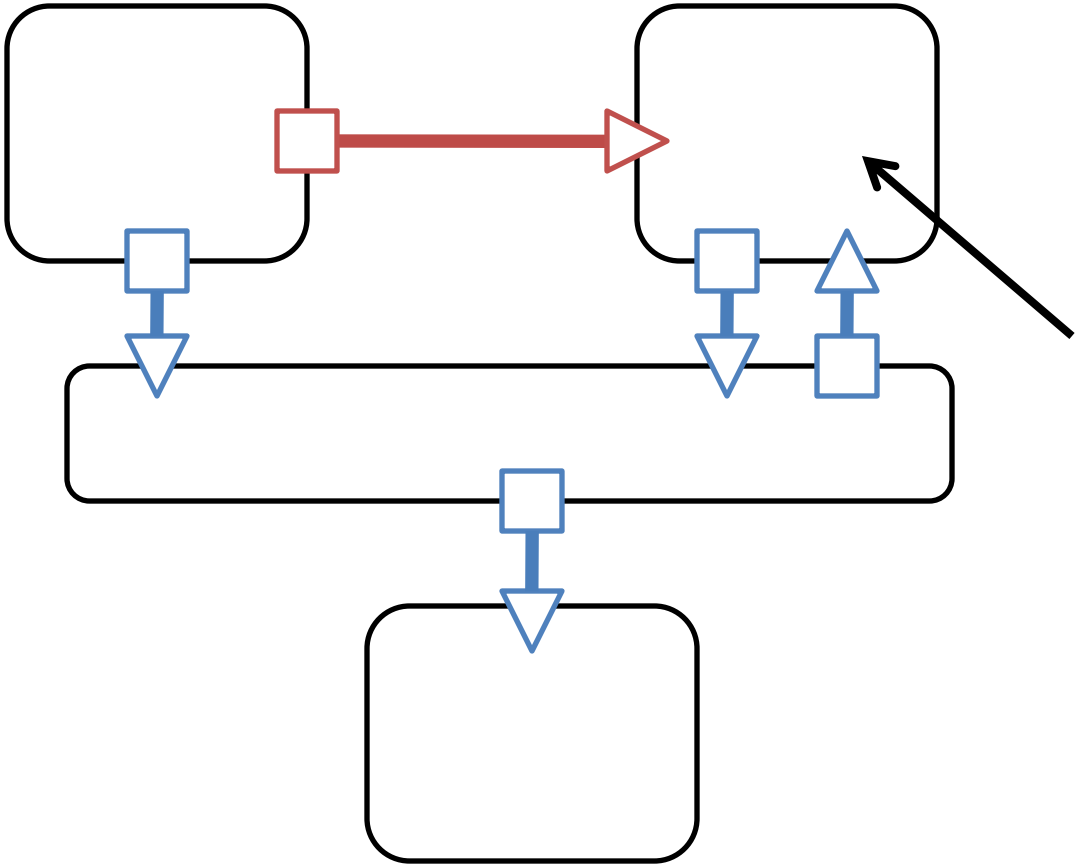
Target port

Initiator port

Data interface

Interrupt interf.

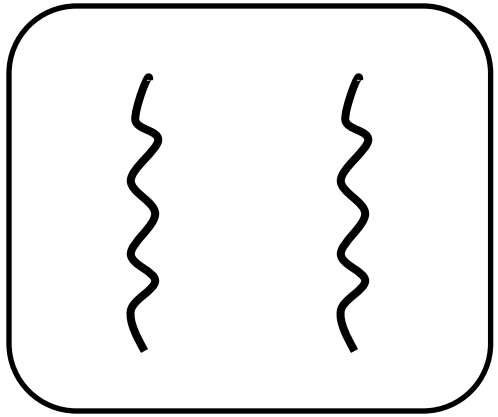
Components



What's inside?

SystemC

- Processes (C++ code)



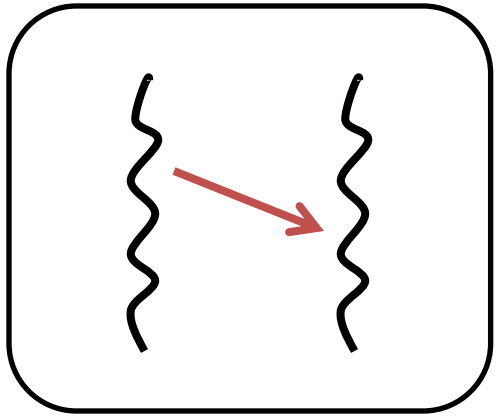
Ready

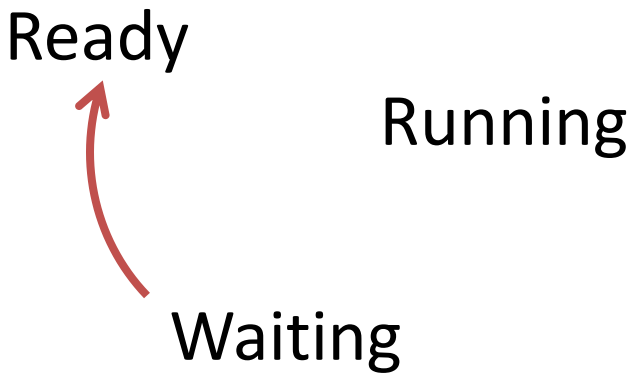
Running

Waiting

SystemC

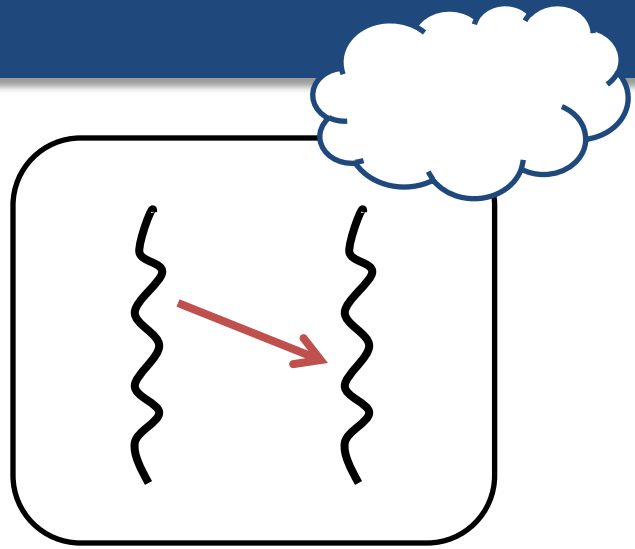
- Processes (C++ code)
- Events

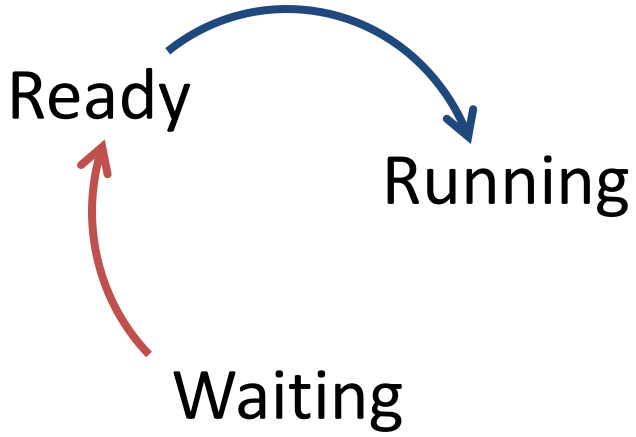




SystemC

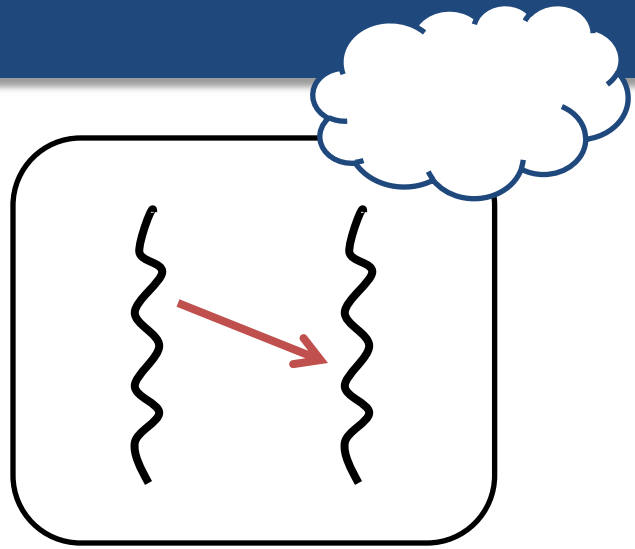
- Processes (C++ code)
- Events
- Scheduler

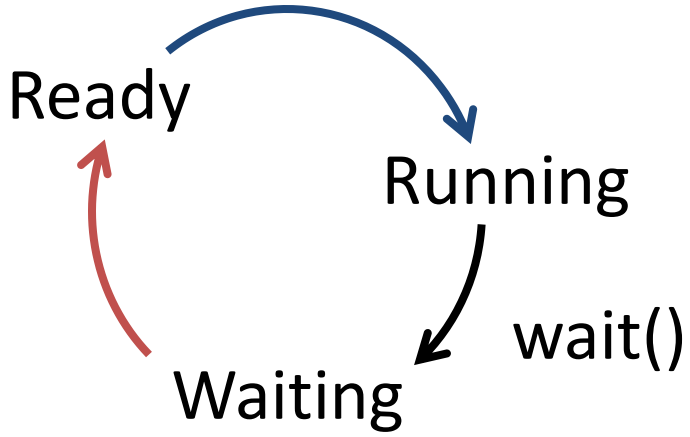




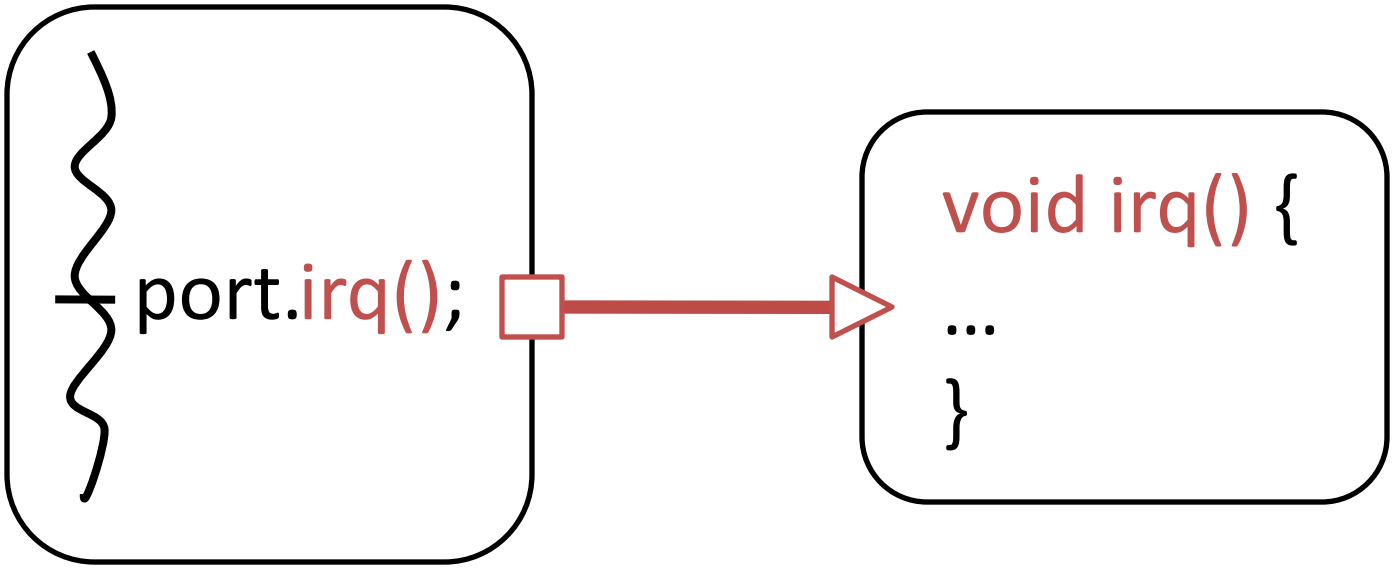
SystemC

- Processes (C++ code)
  - wait()
- Events
- Scheduler

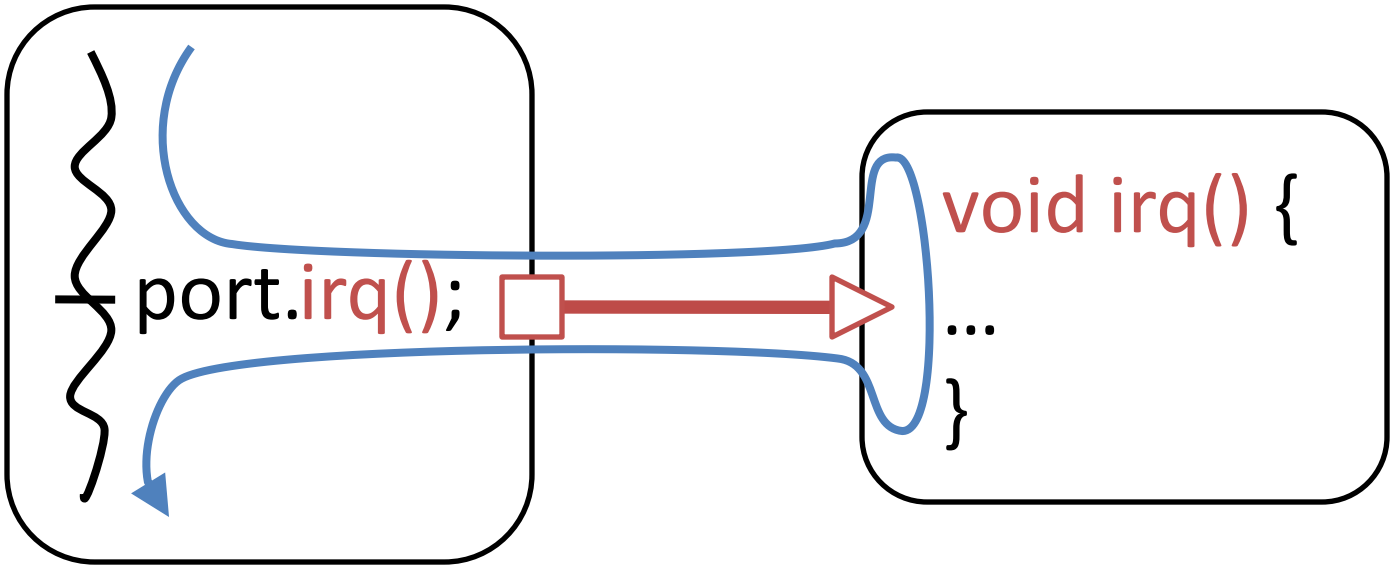




Transactions in SystemC

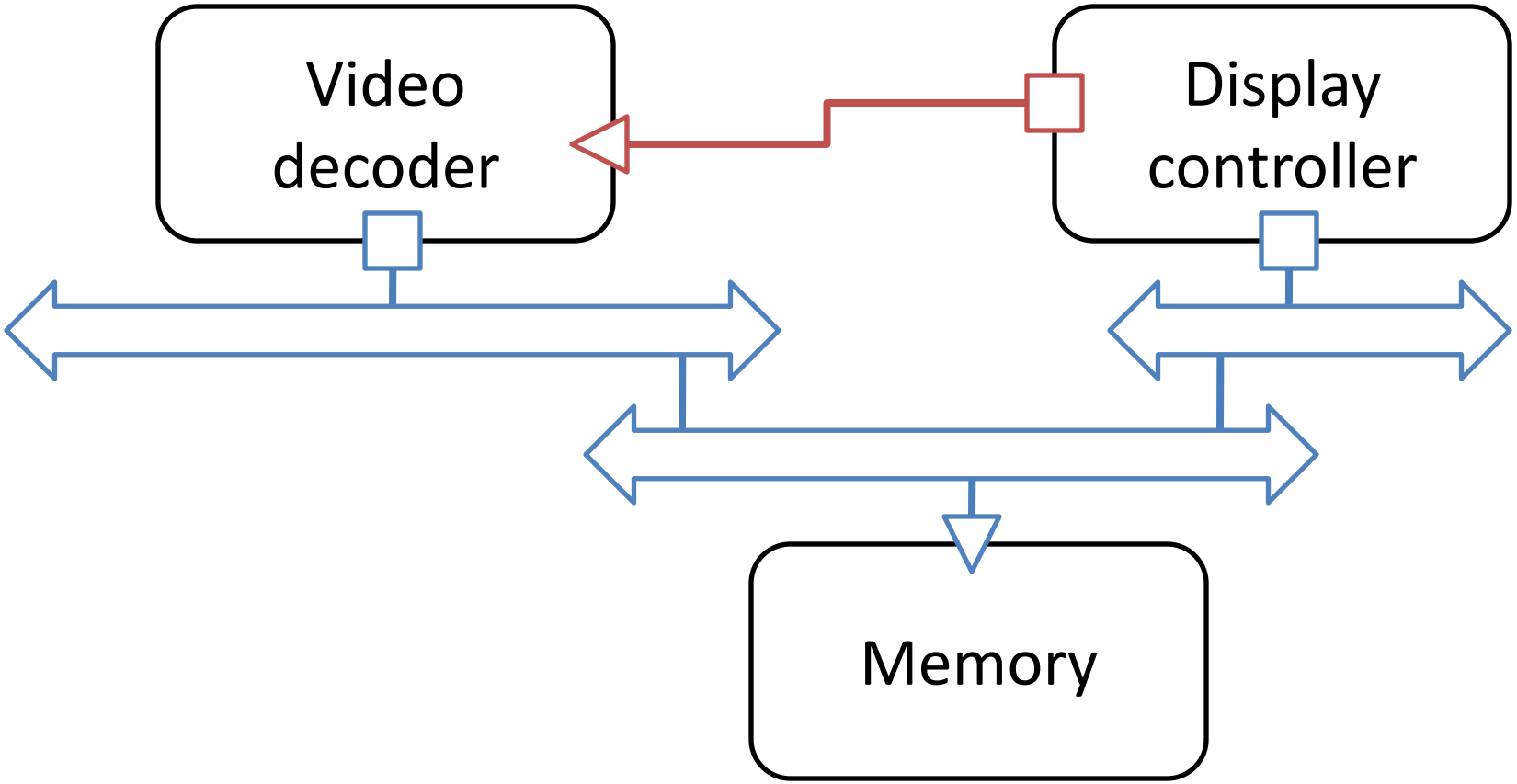


Transactions in SystemC



Interface function calls

An example

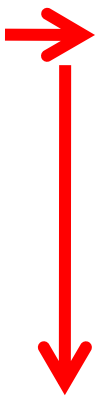


```
Decoder:
  decode(image);
  write(mem, image);
  write(flag, true);
  wait(irq);
```

```
Display:
  do {
     x = read(flag);
  } while(!x);
  img = read(mem);
  display(img);
  irq();
```

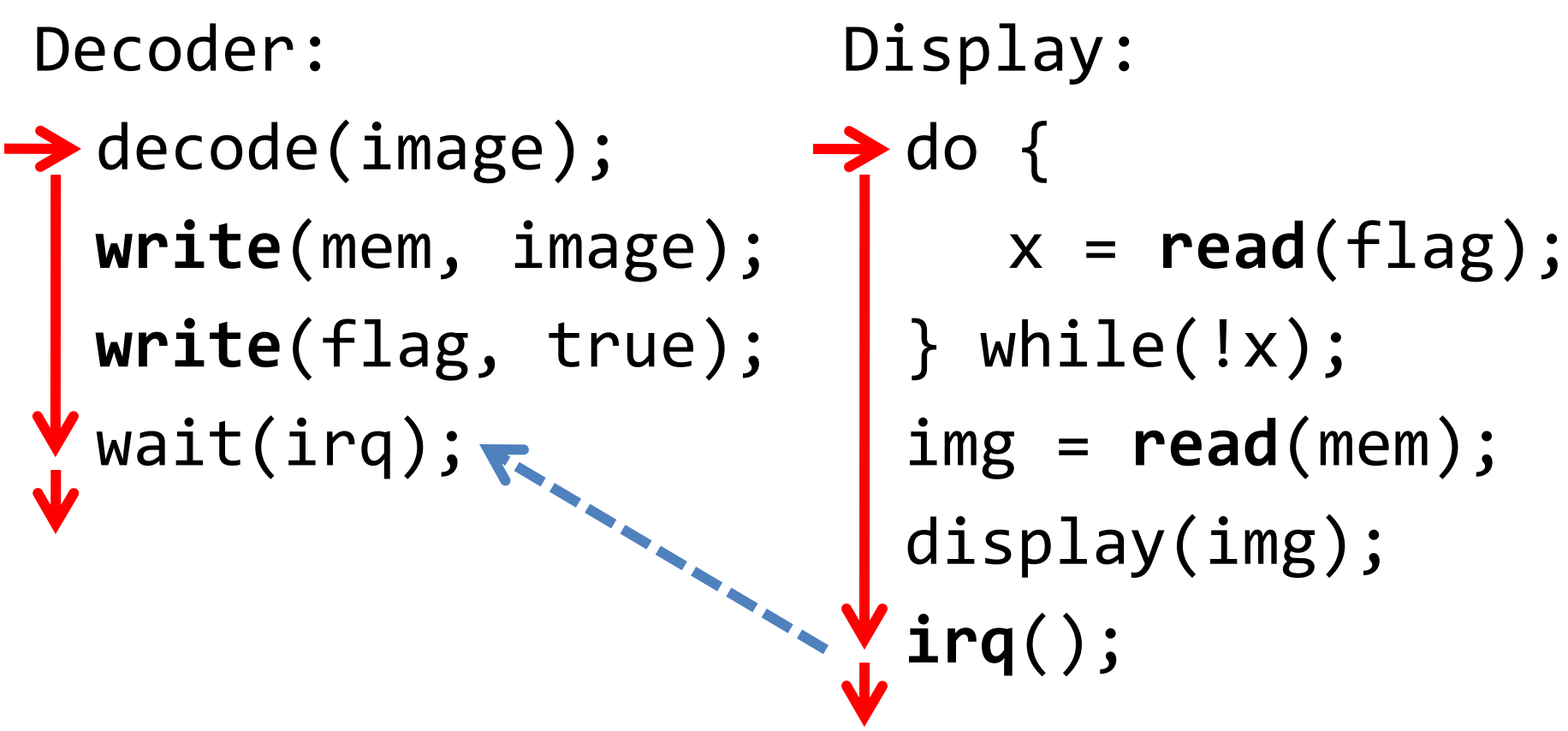
```
Decoder:
  decode(image);
  write(mem, image);
  write(flag, true);
  wait(irq);
```

```
Display:
  do {
     x = read(flag);
  } while(!x);
  img = read(mem);
  display(img);
  irq();
```



```
Decoder:
  decode(image);
  write(mem, image);
  write(flag, true);
  wait(irq);
```

```
Display:
  do {
     x = read(flag);
  } while(!x);
  img = read(mem);
  display(img);
  irq();
```


Decoder:
decode(image);
write(mem, image);
write(flag, true);
wait(irq);
Display:
do {
x = read(flag);
} while(!x);
img = read(mem);
display(img);
irq();

```
Decoder:
  decode(image);
  write(mem, image);
  write(flag, true);
  wait(irq);
```

```
Display:
→ do {
      x = read(flag);
  } while(!x);
  img = read(mem);
  display(img);
  irq();
```

```
Decoder:
  decode(image);
  write(mem, image);
  write(flag, true);
  wait(irq);
```

```
Display:
  do {
     x = read(flag);
  } while(!x);
  img = read(mem);
  display(img);
  irq();
```

yield()

- Like “wait()” but without saying what it is waiting for
- The scheduler may choose the same process again

```
Decoder:
  decode(image);
  write(mem, image);
  write(flag, true);
  wait(irq);
```

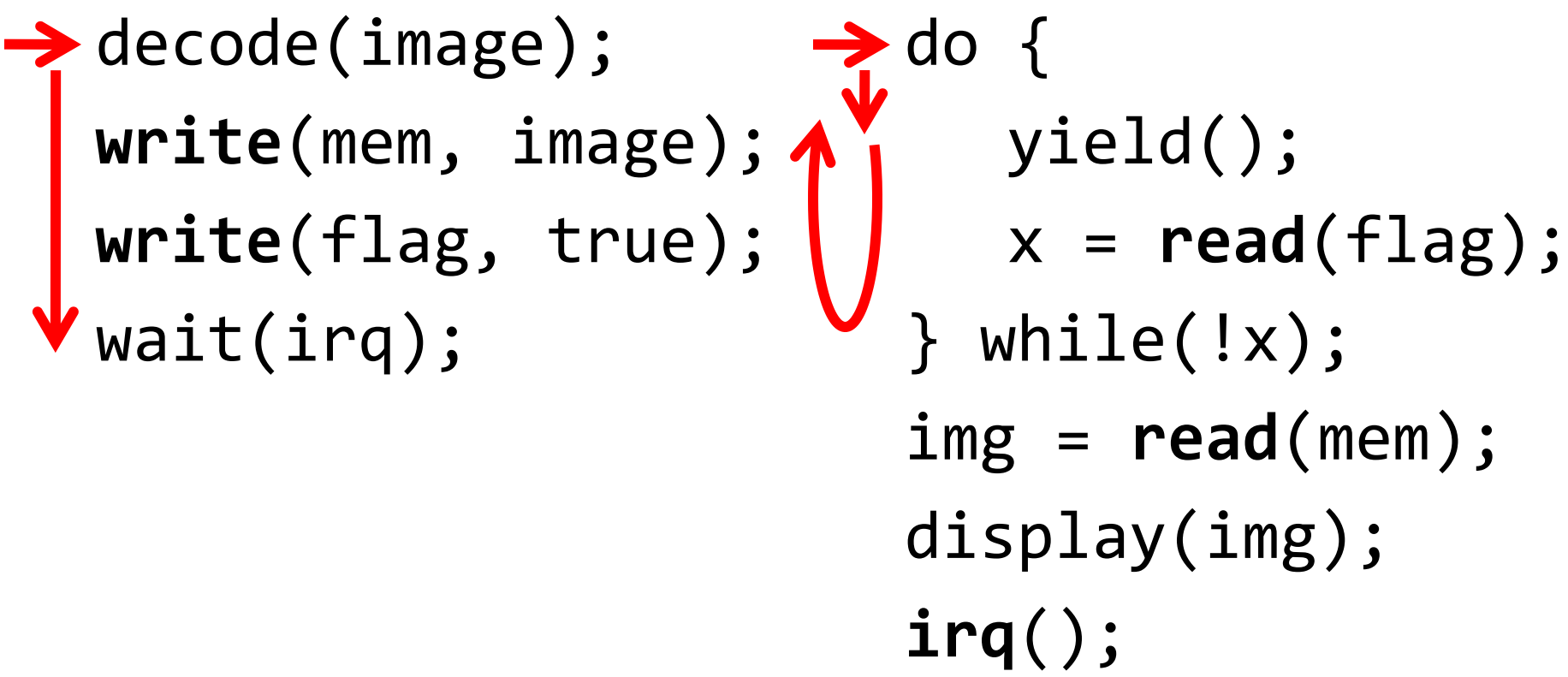
```
Display:
  do {
     yield();
     x = read(flag);
     yield();
  } while(!x);
  img = read(mem);
  display(img);
  irq();
```

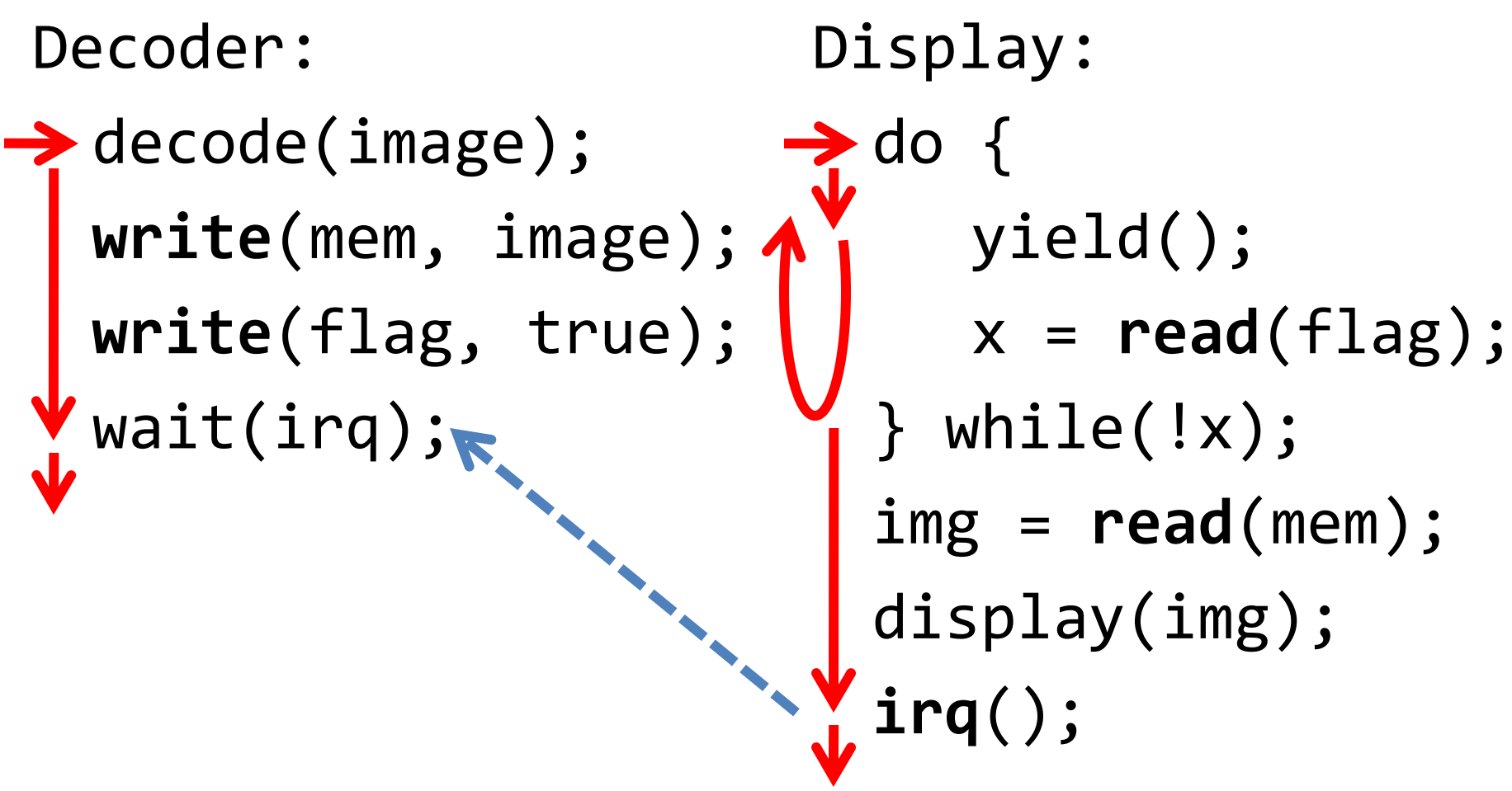
```
Decoder:
  decode(image);
  write(mem, image);
  write(flag, true);
  wait(irq);
```

```
Display:
  do {
     yield();
     x = read(flag);
  } while(!x);
  img = read(mem);
  display(img);
  irq();
```

```
Decoder:
  decode(image);
  write(mem, image);
  write(flag, true);
  wait(irq);
```

```
Display:
  do {
     yield();
     x = read(flag);
  } while(!x);
  img = read(mem);
  display(img);
  irq();
```


Decoder:
decode(image);
write(mem, image);
write(flag, true);
wait(irq);
Display:
do {
yield();
x = read(flag);
} while(!x);
img = read(mem);
display(img);
irq();


Decoder:
decode(image);
write(mem, image);
write(flag, true);
wait(irq);
Display:
do {
yield();
x = read(flag);
} while(!x);
img = read(mem);
display(img);
irq();

Observations

- “yield()”
  1. Avoids livelocks by possibly giving other processes a chance to run
  2. Finds bugs (next slides)

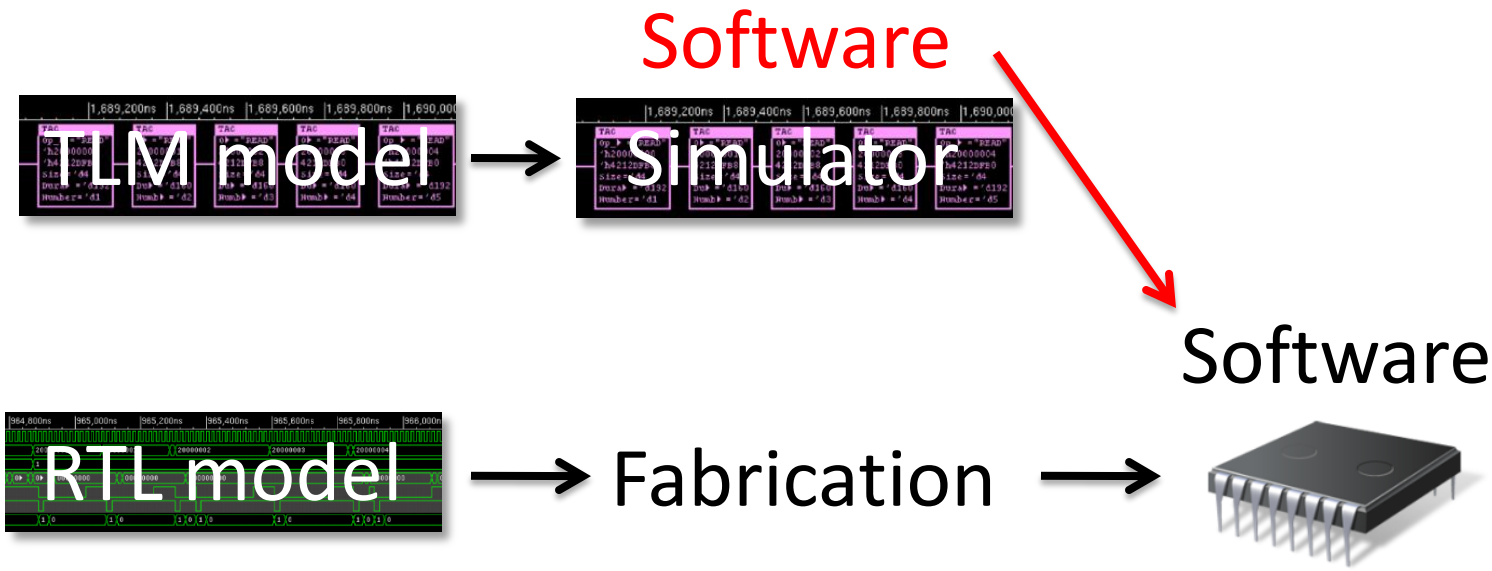
```
Decoder:
  decode(image);
  write(mem, image);
  write(flag, true);
  wait(irq);
```

```
Display:
  do {
     yield();
     x = read(flag);
  } while(!x);
  img = read(mem);
  display(img);
  irq();
```

```
Decoder:
  decode(image);
  write(flag, true);
  write(mem, image);
  wait(irq);
```

```
Display:
  do {
     yield();
     x = read(flag);
  } while(!x);
  img = read(mem);
  display(img);
  irq();
```

Design flow


Software
TLM model
Simulator
Software
RTL model
Fabrication

```
Decoder:
  decode(image);
  write(flag, true);
  yield();
  write(mem, image);
  wait(irq);
```

```
Display:
  do {
     yield();
     x = read(flag);
  } while(!x);
  img = read(mem);
  display(img);
  irq();
```

```
Decoder:
  decode(image);
  write(mem, image);
  yield();
  write(flag, true);
  wait(irq);
```

```
Display:
  do {
     x = read(flag);
     yield();
  } while(!x);
  img = read(mem);
  display(img);
  yield();
  irq();
```

Conclusions

- A minimum amount of yielding makes a certain platform work
- A bit more is needed to
  - make any platform work
  - finding bugs
- Essential for component based approach!

Approaches at ST

- TAC1
  - yield() between any two accesses
  - Poor performance

Approaches at ST

- TAC2
  - Tag addresses “synchro” () or “data”
  - yield() after any “synchro”
  - Very good performance

```
Decoder:
  decode(image);
  write(mem, image);
  write(flag, true);
  wait(irq);
```

```
Display:
  do {
    x = read(flag);
  } while(!x);
  img = read(mem);
  display(img);
  irq();
```

```
Decoder:
  decode(image);
  write(mem, image);
  write(flag, true);
  yield();
  wait(irq);
```

```
Display:
  do {
    x = read(flag);
    yield();
  } while(!x);
  img = read(mem);
  display(img);
  irq();
```

```
Decoder:
  decode(image);
  write(mem, image);
  yield();
  write(flag, true);
  yield();
  wait(irq);
```

```
Display:
  do {
    x = read(flag);
    yield();
  } while(!x);
  img = read(mem);
  display(img);
  yield();
  irq();
```

State-of-the-art

- TAC2.5
  - Experimental results by Jérôme Cornet
  - Tag addresses “synchro” () or “data”
  - yield()
    - before and after any “synchro”
    - before irq’s

```
Decoder:
  decode(image);
  write(mem, image);
  write(flag, true);
  wait(irq);
```

```
Display:
  do {
    x = read(flag);
  } while(!x);
  img = read(mem);
  display(img);
  irq();
```

```
Decoder:
  decode(image);
  write(mem, image);
  yield();
  write(flag, true);
  yield();
  wait(irq);
```

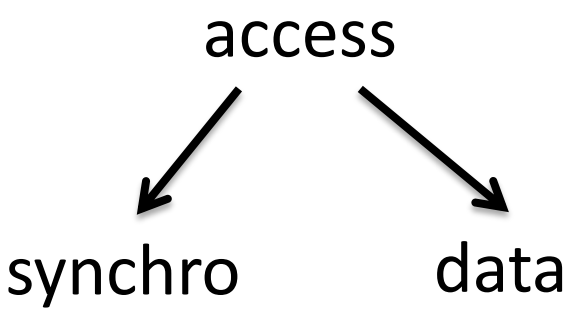
```
Display:
  do {
    yield();
    x = read(flag);
    yield();
  } while(!x);
  img = read(mem);
  display(img);
  yield();
  irq();
```

Memory consistency contracts

- Contract between processor and software in a multiprocessor system
  - Assume: software follows guidelines
  - Guarantee: memory stays consistent
    - Cache flushes, out-of-order execution,…

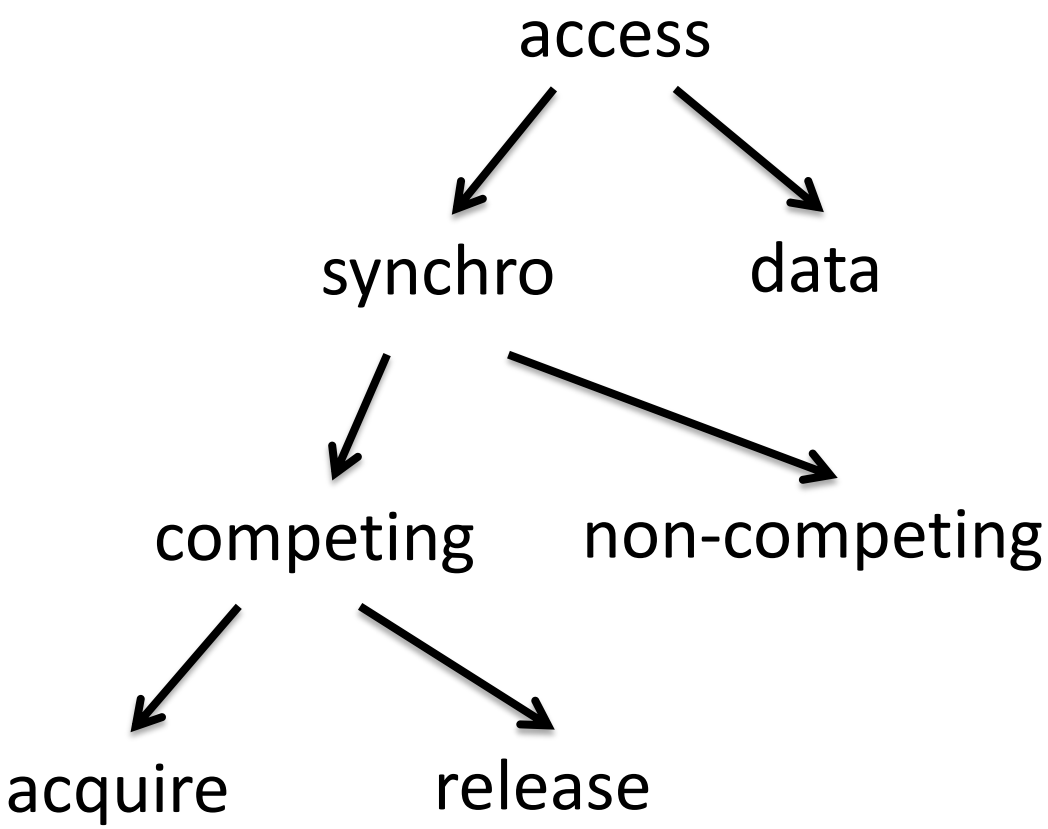
Memory consistency contracts

- Guidelines:



Memory consistency contracts

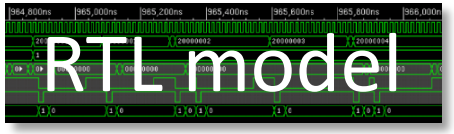
- Guidelines:



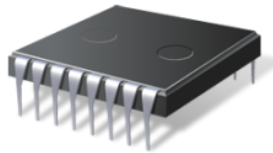
Design flow

Seen as specification

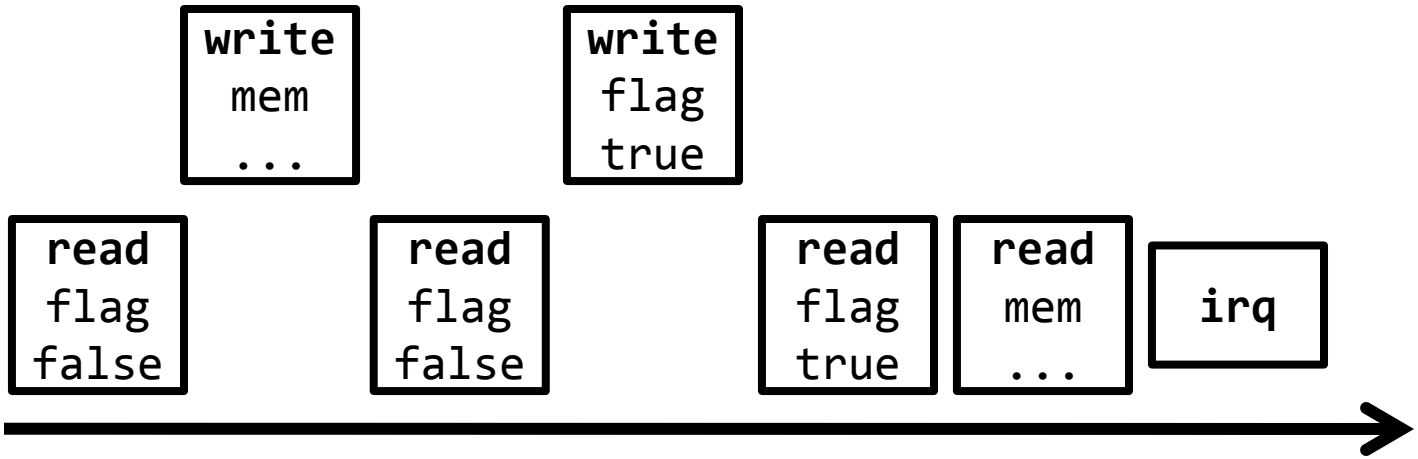
Valid implementation?



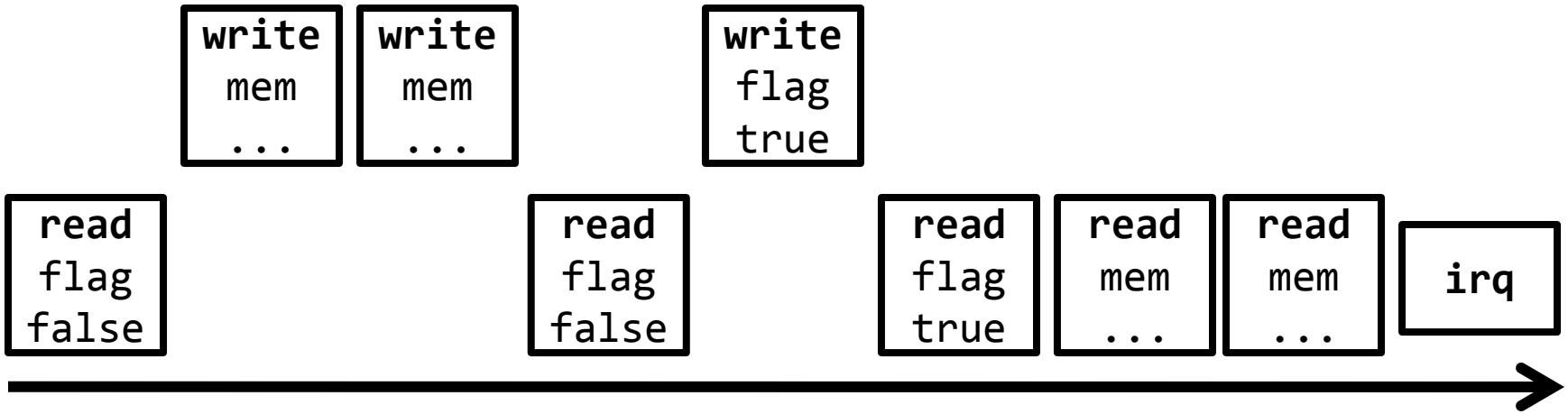
→ Fabrication →



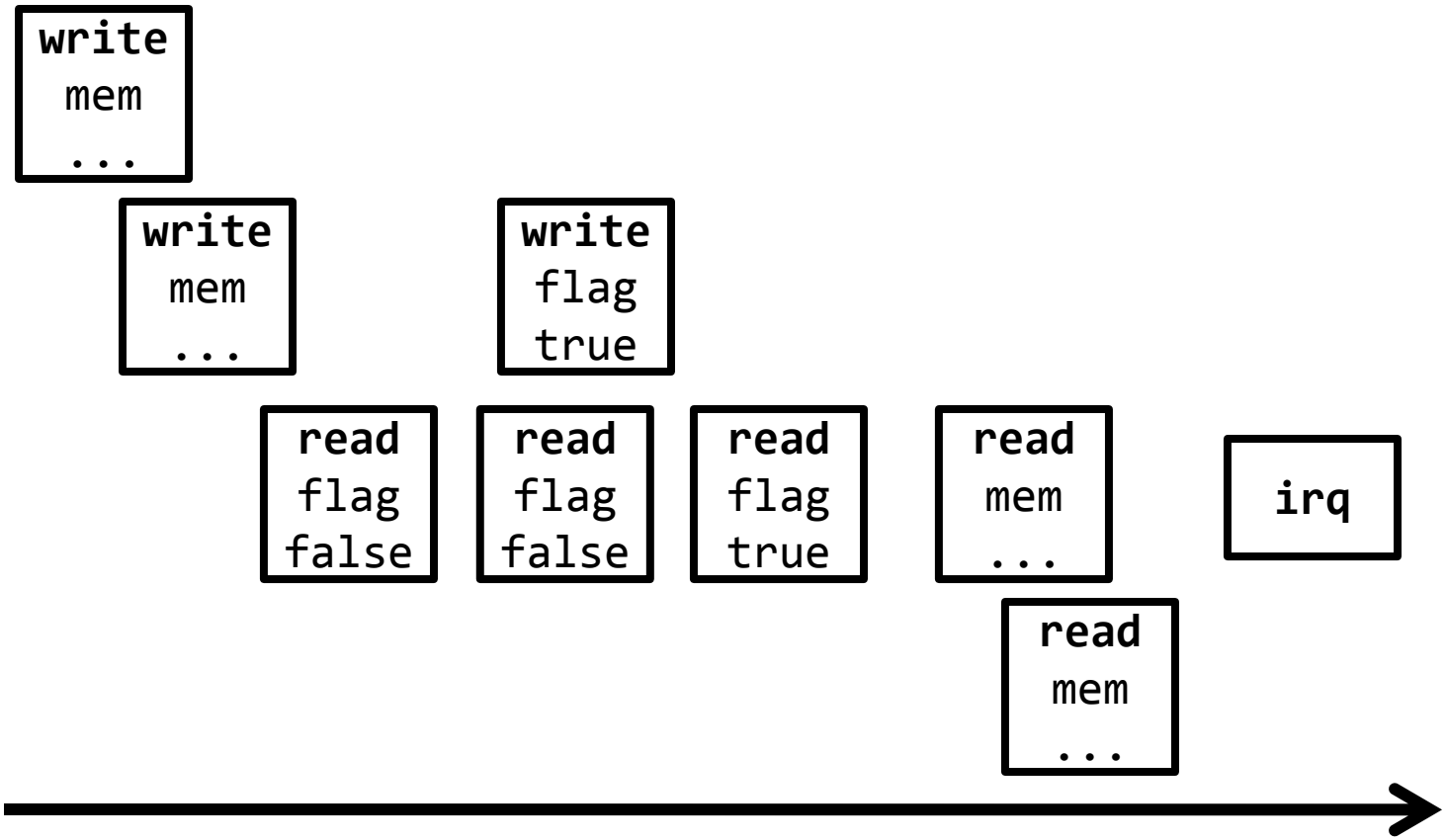
From TLM…


write
mem
...
write
flag
true
read
flag
false
read
flag
false
read
flag
true
read
mem
...
irq

...to RTL

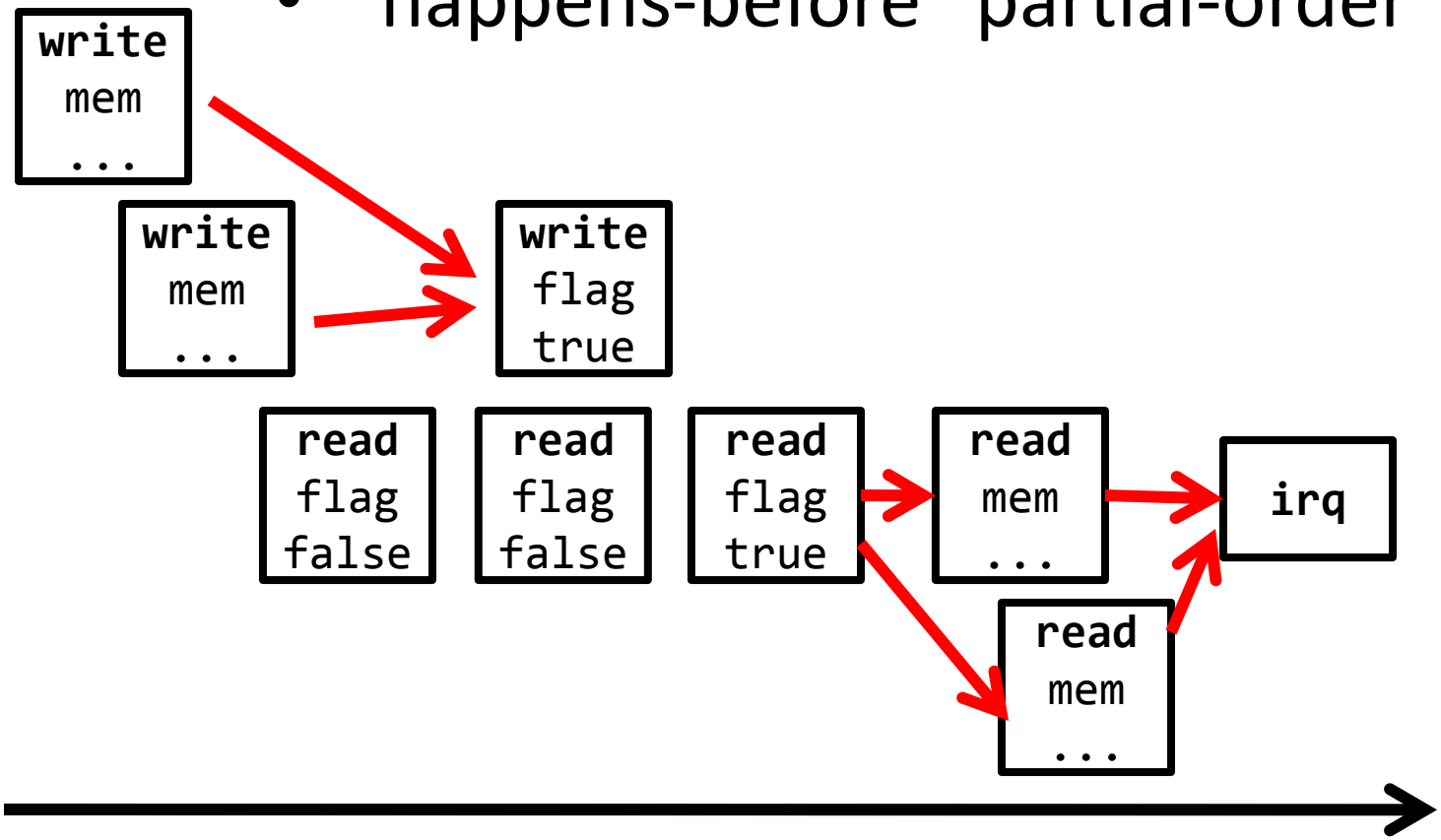

write
mem
...
write
mem
...
write
flag
true
read
flag
false
read
flag
false
read
flag
true
read
mem
...
read
mem
...
irq

...to RTL

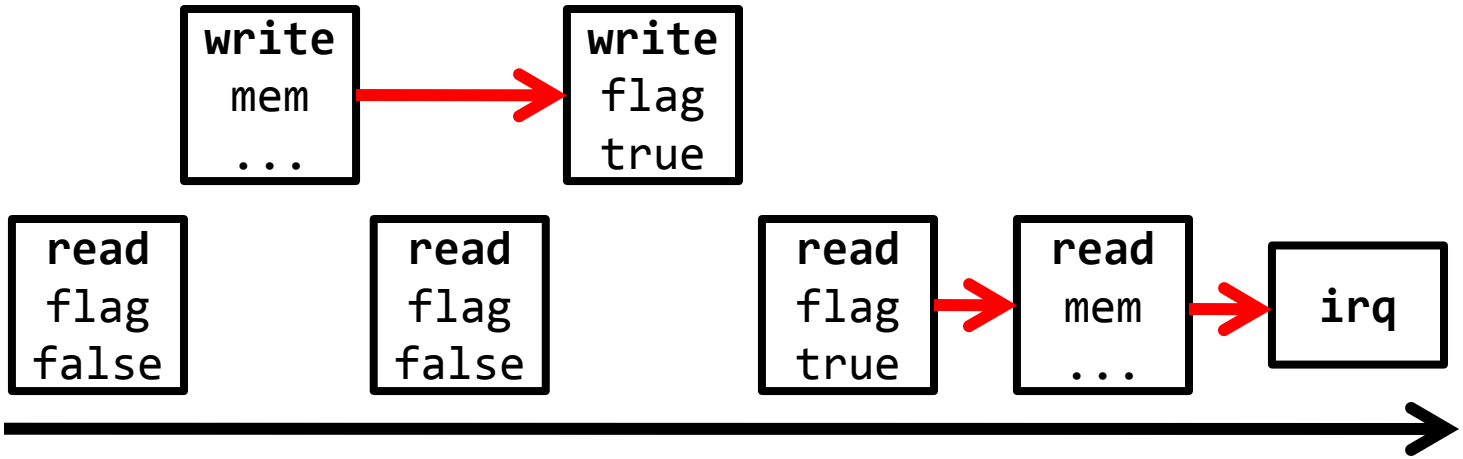

write
mem
...
write
mem
...
write
flag
true
read
flag
false
read
flag
false
read
flag
true
read
mem
...
irq
read
mem
...

And from RTL…

- "happens-before" partial-order



Back to TLM…


write
mem
...
write
flag
true
read
flag
false
read
flag
false
read
flag
true
read
mem
...
irq

Understand yield

- Link with “happens-before” partial-order

```
Decoder:
  decode(image);
  write(mem, image);
  write(flag, true);
  wait(irq);
```

```
Display:
  do {
     x = read(flag);
  } while(!x);
  img = read(mem);
  display(img);
  irq();
```

Future work

- Import memory consistency contracts to TLM
  - Formal justification for “synchro”
  - Better understanding helps placing tags
- Multi abstraction-level components
  - “happens-before” order must be maintained by valid implementations (contracts?)

Components and abstraction levels for system-on-chip

Giovanni Funchal
STMicroelectronics
Verimag

Synchron'2008