6.189 IAP 2007

Lecture 14

Synthesizing Parallel Programs

Synthesizing parallel programs
(or borrowing some ideas from hardware design)

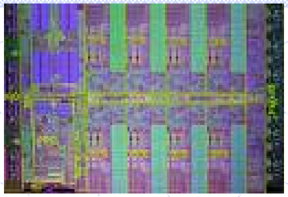
Arvind
Computer Science & Artificial Intelligence Lab.
Massachusetts Institute of Technology

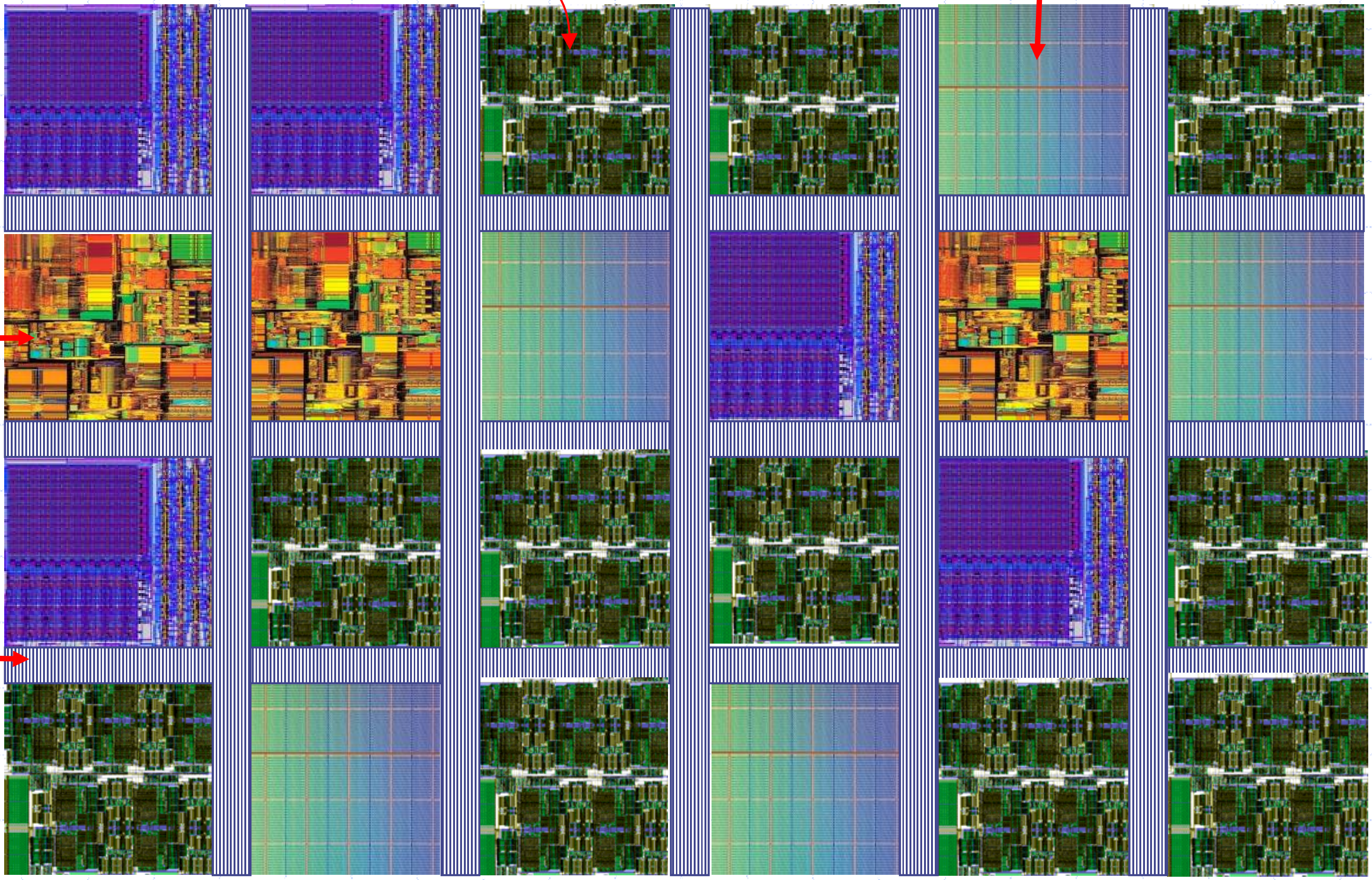
6.189

January 24, 2007

SoC Trajectory:
multicores, heterogeneous, regular, ...

IBM Cell Processor





Can we rapidly produce high-quality chips and surrounding systems and software?

Plan for this talk

- My old way of thinking (up to 1998)
  - “Where are my threads?”
  - Not necessarily wrong

- My new way of thinking (since July)
  - “Parallel program module as a resource”
  - Not necessarily right

Connections with transactional programming, though obvious, not fully explored yet

Acknowledgement: Nirav Dave

Only reason for parallel programming used to be performance

- This made programming very difficult
  - Had to know a lot about the machine
  - Codes were not portable – endless performance tuning on each machine
  - Parallel libraries were not composable
  - Difficult to deal with heap structures and memory hierarchy
  - Synchronization costs were too high to exploit fine-grain parallelism

How to exploit 100s of threads from software?

Implicit Parallelism

- Extract parallelism from programs written in sequential languages
  - Lot of research over four decades – limited success
- Program in functional languages which may not obscure parallelism in an algorithm

If the algorithm has no parallelism then forget it

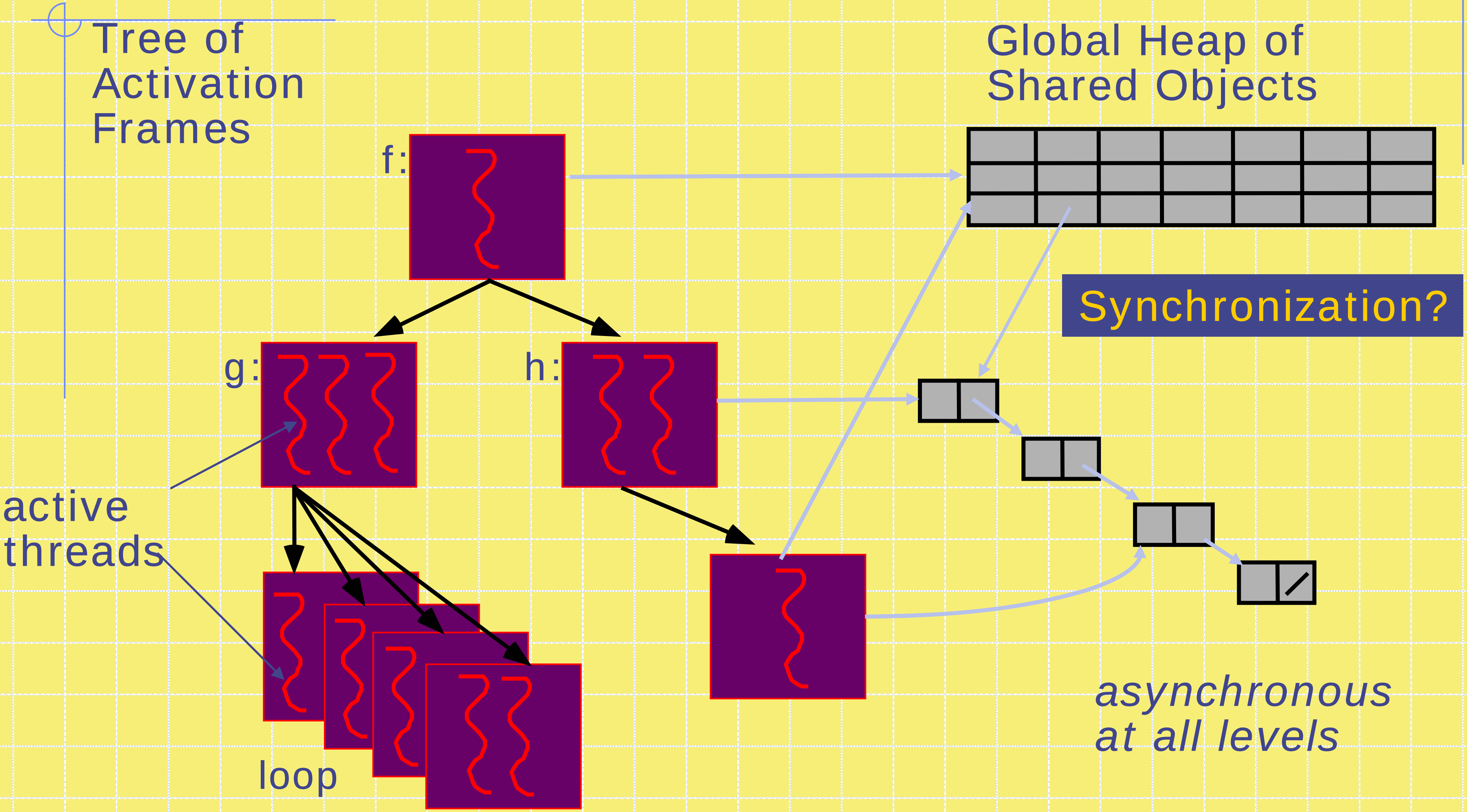
If parallelism can't be detected automatically ...

Design/use new explicitly parallel programming models ...

Works well but not general enough

- High-level
  - Data parallel: *Fortran 90, HPF, ...*
  - Multithreaded: *Id, pH, Cilk,..., Java*
- Low-level
  - Message passing: *PVM, MPI, ...*
  - Threads & synchronization: *Forks & Joins, Locks, Futures, ...*

Fully Parallel, Multithreaded Model



Efficient mappings on architectures proved difficult

My unrealized dream

A time when Freshmen will be taught sequential programming as a special case of parallel programming

Has the situation changed?

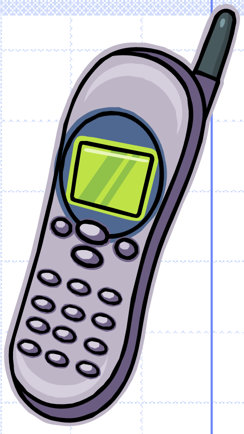
- Yes
  - Multicores have arrived
  - Even Microsoft wants to exploit parallelism
  - Explosion of cell phones
  - Explosion of game boxes

Freshmen are going to be hacking game boxes and cell phones

It is all about parallelism now!

now ...

Cell phone



- Mine sometimes misses a call when I am surfing the web
  - To what extent the phone call software should be aware of web surfing software, or vice versa?
  - Is it merely a scheduling issue?
  - Is it a performance issue?

Sequential “modules” are often used in concurrent environments with unforeseen consequences

New Goals

Synthesis as opposed to *Decomposition*

Know how to do this

- A method of designing and connecting modules such that the functionality and performance are predictable
  - Must facilitate natural descriptions of concurrent systems
- A method of refining individual modules into hardware or software for SoCs
- A method of mapping such designs onto "multicores"
  - Time multiplexing of resources complicates the problem

A hardware inspired methodology for “synthesizing” parallel programs

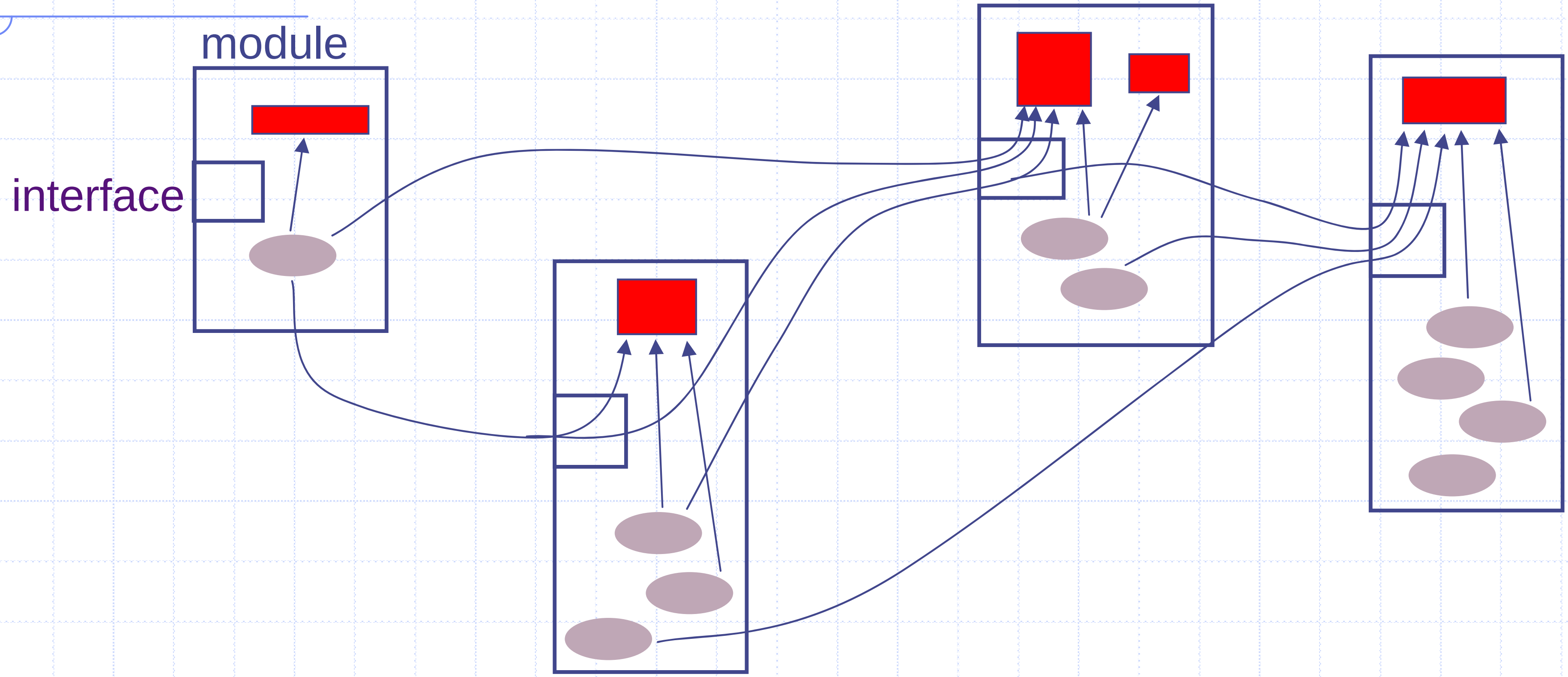
- Rule-based specification of behavior (Guarded Atomic Actions)
  - Lets you think one *rule* at a time
- Composition of modules with guarded interfaces

Bluespec

Unity – late 80s
Chandy & Misra

- Some examples:
  - GCD
  - Airline reservation
  - Video codec: H.264
  - Inserting in an ordered list

Bluespec: State and Rules organized into *modules*



All *state* (e.g., Registers, FIFOs, RAMs, ...) is explicit.

Behavior is expressed in terms of atomic actions on the state:

Rule: condition ➔ action

Rules can manipulate state in other modules only *via* their *interfaces*.

Execution model

Repeatedly:

- Select a rule to execute
- Compute the state updates
- Make the state updates

Highly non-deterministic

Primitives are provided to control the selection

Example: Euclid's GCD

A GCD program

GCD(x, y) = ***if*** y = 0 ***then*** x
elseif x>y ***then*** GCD(y, x)
else GCD(x, y-x)

Execution

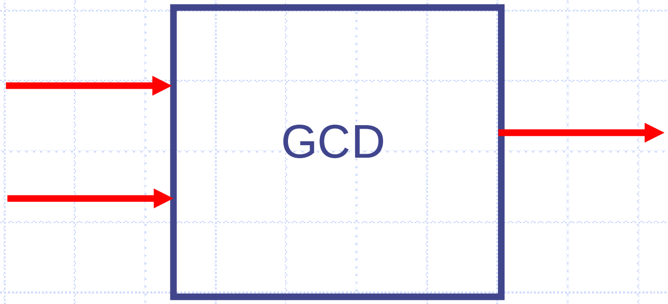
GCD(6, 15) $\Rightarrow$ GCD(6, 9) $\Rightarrow$ GCD(6, 3) $\Rightarrow$

GCD(3, 6) $\Rightarrow$ GCD(3, 3) $\Rightarrow$ GCD(3, 0) $\Rightarrow$ 3

What does this program mean in a concurrent setting ?

GCD(623971, 150652) + GCD(1543276, 9760552)

Suppose we want to build a GCD machine (i.e., IP module)



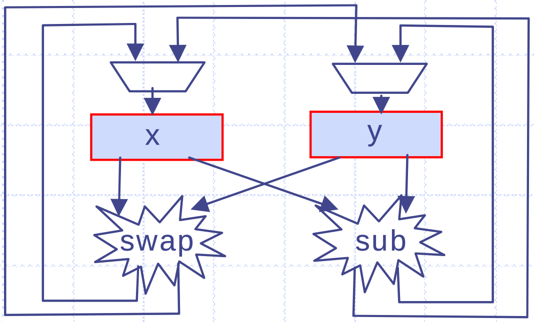
GCD as a resource

- Parallel invocations?
  - Recursive calls vs Independent calls
- Does the answer come out immediately? In predictable time?
- Can the machine be shared?
- Can it be pipelined, i.e., accept another input before the first one has produced an answer?

These questions arise naturally in hardware design

But these questions are equally valid in a parallel software setting

GCD in Bluespec



Synthesized hardware

```
module mkGCD
    x <- mkReg(0);
    y <- mkReg(0);

    rule swap when ((x > y) & (y != 0)) ==>
            x := y | y := x

    rule subtract when ((x <= y) & (y != 0)) ==>
            y := y – x

    method start(a,b) when (y==0) ==>
            x := a | y := b

    method result() when (y==0) ==> return (x)
end
```

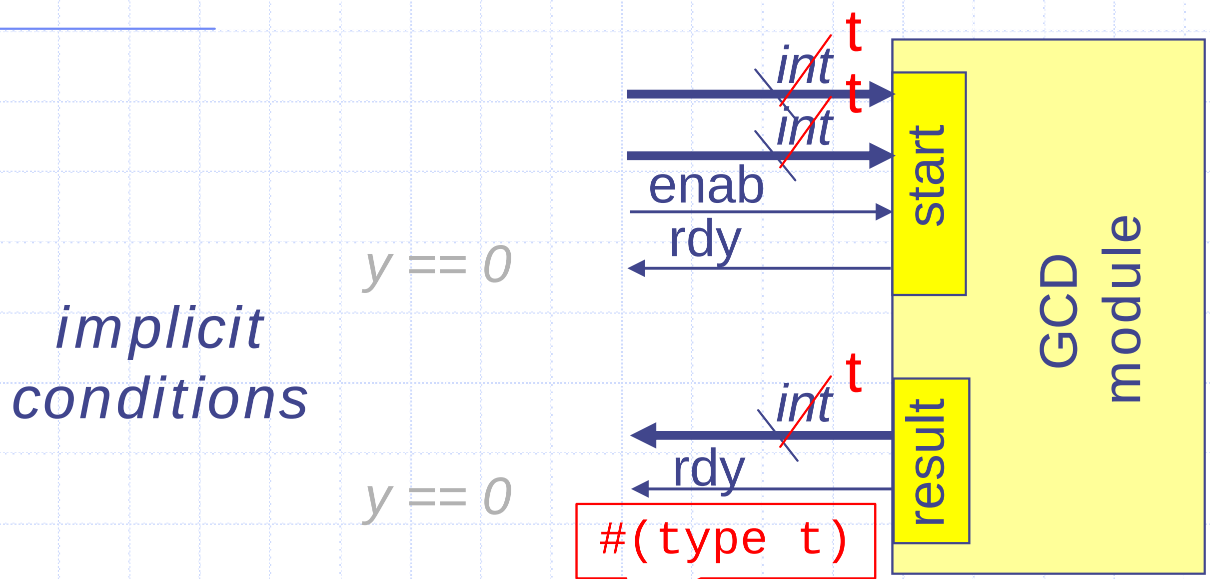
State

Internal behavior

External interface

What happened to the recursive calls?

GCD Hardware Module



In a GCD call t could be
`Int#(32)`,
`UInt#(16)`,
`Int#(13)`, ...

interface I_GCD;
    method Action start (~~int~~t a, ~~int~~t b);
    method ~~int~~t result();
endinterface

- The module can easily be made polymorphic
- Many different implementations, *including pure software ones*, can provide the same interface

`module mkGCD (I_GCD)`

The Bluespec Language

Bluespec: A Language of Atomic Actions

A program is a collection of instantiated modules m_1 ; m_2 ; ...

Module ::= Module name
[State variable r]
[Rule R a]
[Action method g (x) = a]
[Read method f (x) = e]

```
a ::=  r := e
     | if e then a      <- Conditional action
     | a | a            <- Parallel Composition
     | a ; a            <- Sequential Composition
     | (t = e in a)
     | m.g(e)           <- Method call
     | a when e         <- Guarded action

e ::=  r | c | t
     | Op(e , e)
     | e ? e : e
     | (t = e in e)
     | m.f(e)
     | e when e
```

Guards vs If's

- Guards affect the surroundings

(a1 when p1) | a2 ==> (a1 | a2) when p1

- Effect of an "if" is local

(if p1 then a1) | a2 ==> if p1 then (a1 | a2) else a2

p1 has no effect on a2

Airline Reservation

Example: Airline reservation

a problem posed by Jayadev Misra

- Ask quotes from two airlines
  - If any one quotes below $300, buy immediately
  - Buy the lower quote if over $300
  - After one minute buy from whosoever has quoted, otherwise flag error

Express it using threads?

Complicated

Solution is easy to express in Misra's ORC “done” also means “not busy”

Solution in Bluespec

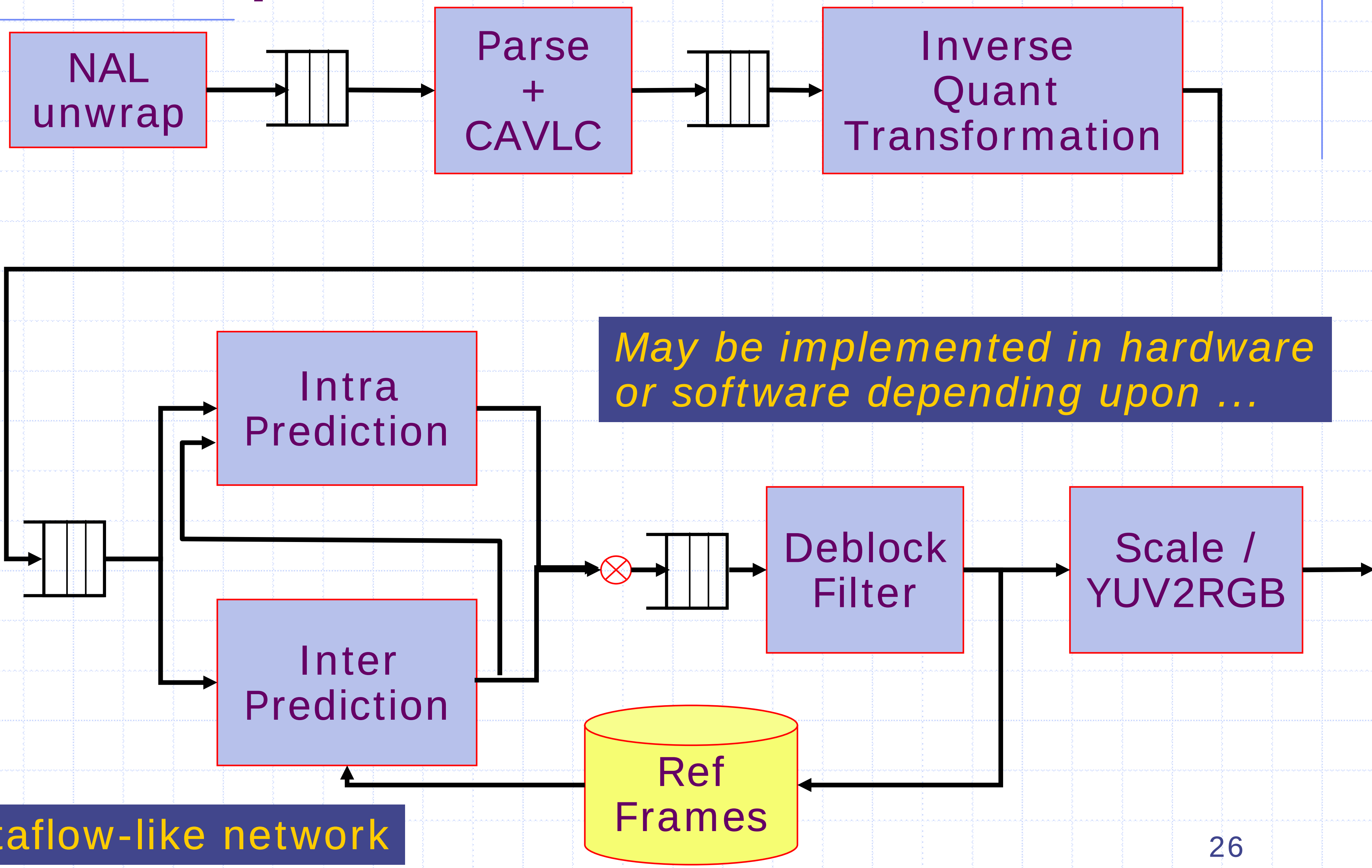
Straightforward

```
module mkGetQuotes();
  define state elements Aquote, Bquote, done, timer
rule pickCheapest when
      !done & (Aquote != INF) & (Bquote != INF) ==>
       (if (Aquote < Bquote) then ticket <- A.purchase(Aquote)
                              else  ticket <- B.purchase(Bquote))
      | (done := True)
  rule getA when !done ==> … // executes when A responds
  rule getB …   rule timeout …   rule timer
  method bookTicket(r) when done ==>
        A.request(r)    | B.request(r)    | done := False
      | Aquote := INF | Bquote := INF | timer :=0

  method getTicket()    when done ==>  return (ticket)
end
```

Video Codec: H.264

Example: H.264 Decoder



May be implemented in hardware or software depending upon ...

A dataflow-like network

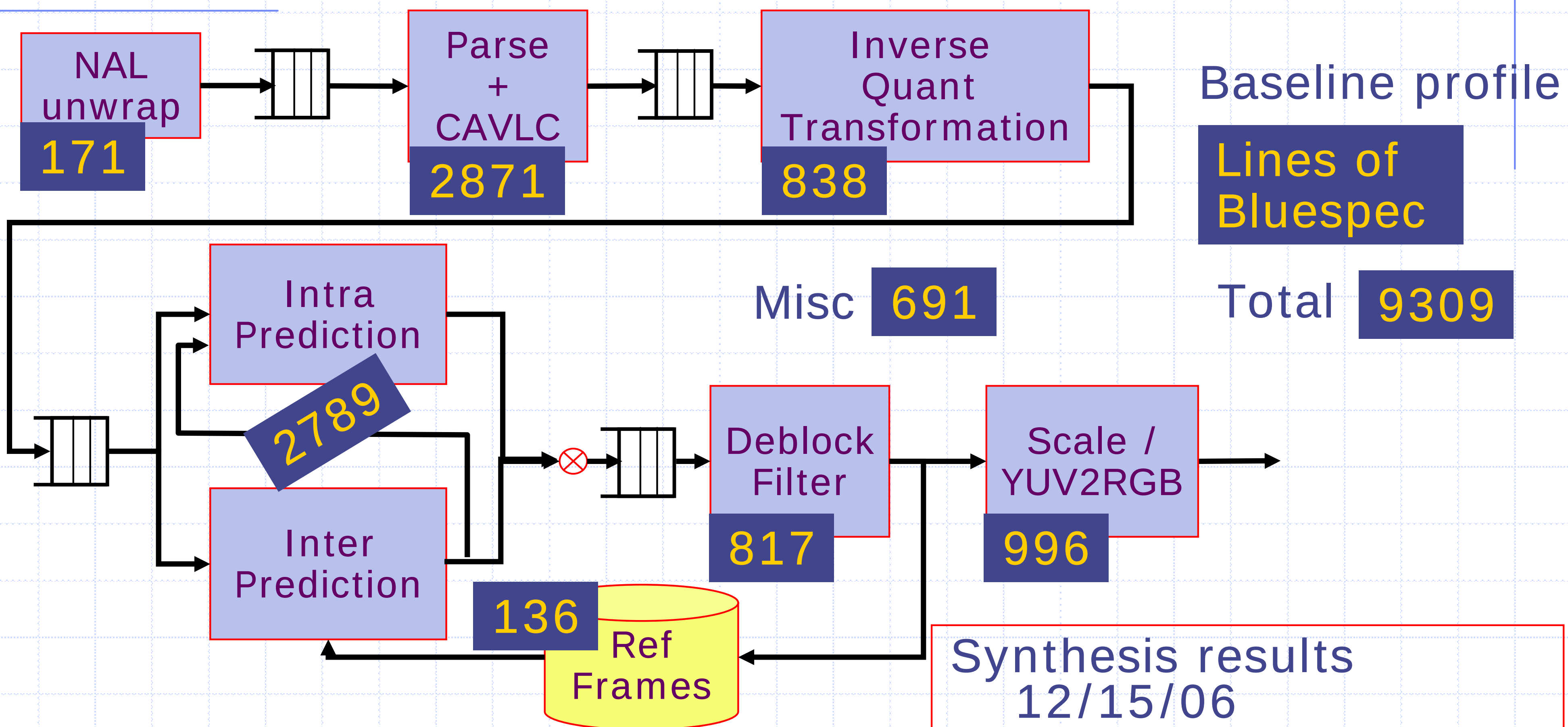
Available codes (*not multithreaded*)

- Reference code
  - 80K lines, awful coding style, slow
- ffmpeg code for Linux
  - 200K lines, mixed with other codecs
- Codes don't reflect the dataflow structure
  - Pointers to data structures are passed around and modified. Difficult to figure out which block is modifying which parts
  - No model of concurrency. Even the streaming aspect gets obscured by the code

The code can be written in a style which will serve both hardware and software communities.

H.264 Decoder in Bluespec

Work in Progress - Chun-Chieh Lin et al



- Any module can be implemented in software
- Each module can be refined separately
- Behaviors of modules are composable
  - Good source code for multicores

Synthesis results 12/15/06

- Decodes 720p@18fps
- Critical path 50Mz
- Area 5.5 mm sq

Takeaway

- Parallel programming should be based on well defined modules and parallel composition of such modules
- Modules must embody a notion of resources, and consequently, sharing and time-multiplexed reuse
- Guarded Atomic Actions and Modules with guarded interfaces provide a solid foundation for doing so

Thanks