High-Level Synthesis Lessons

Understanding structure and appreciating mother-nature

Rajesh Gupta

University of California, San Diego.

ECSI, Darmstadt September 06

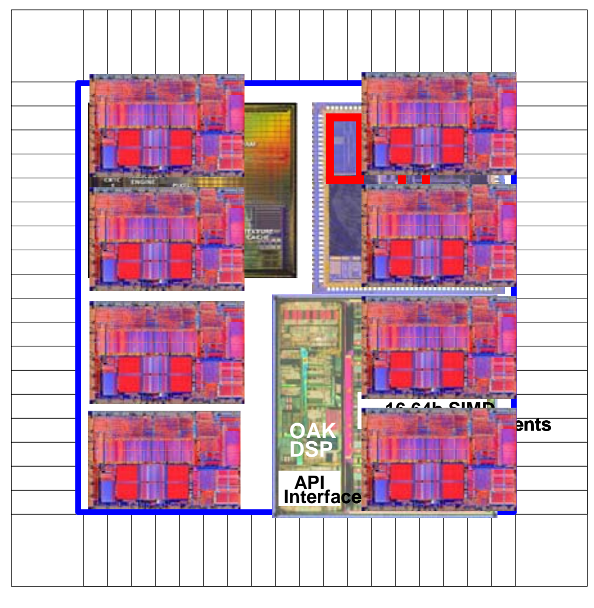
MESL . UCSD . EDU

Two Observations

#1 Today's silicon is a lot about cost and capacity

#2 Silicon architectures matter

- Intrinsic Si efficiency ranges by 10^2-10^3X depending upon computation fabric used (MOPS/W, MOPS/mm^2)
  - MPU: 100 MOPS/W
  - FPGA: 1-2 GOPS/W
  - ASIC: 10-20 GOPS/W

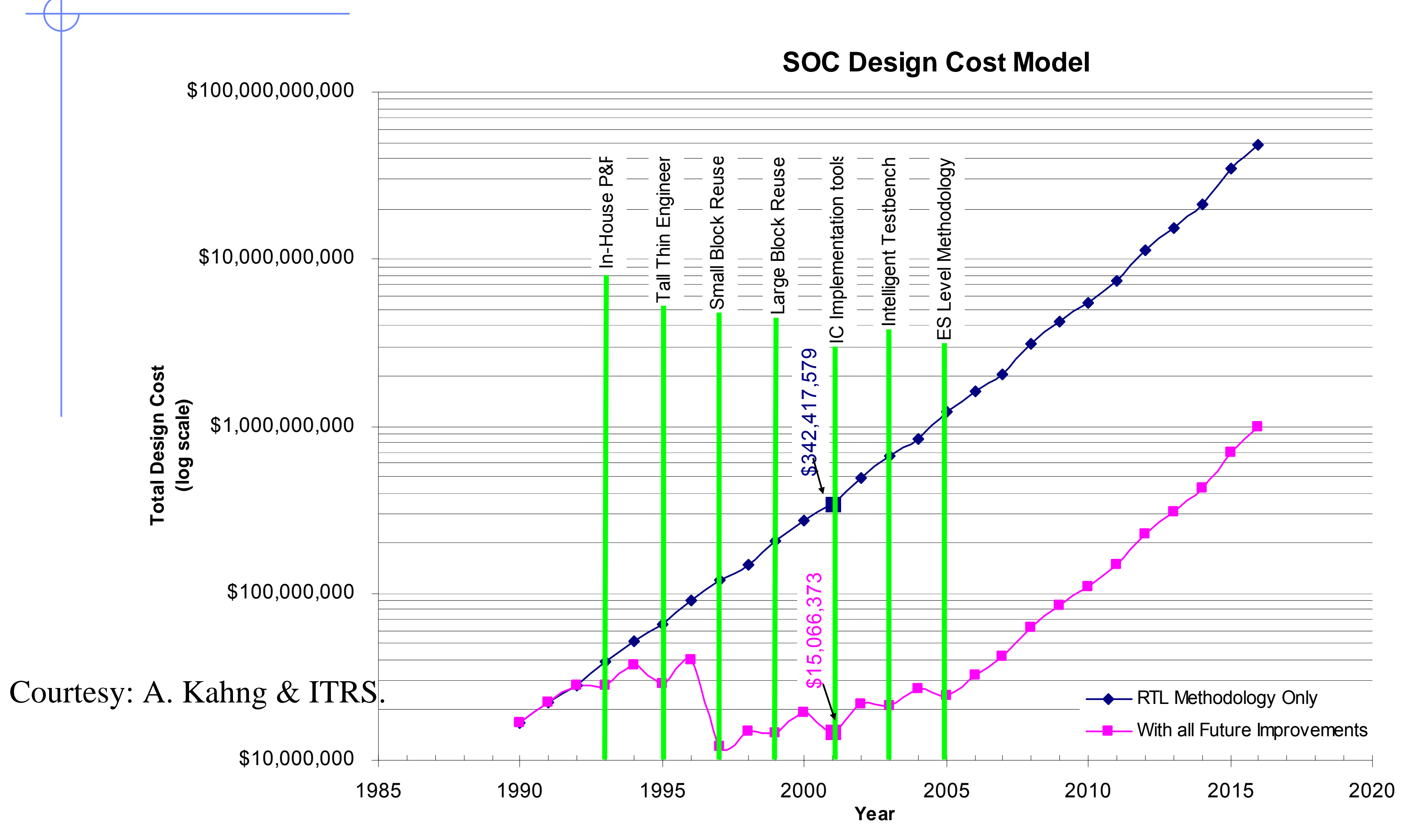


Pad limited die:
200 pins
52 mm^2
>1K dies/wafer
$5/part

➲If done right, there is a space of 100-1000x gain in Silicon efficiency in hardware realization

Power, Reliability bugs when pushing hard on these

But getting there is not cheap



Courtesy: A. Kahng & ITRS.

➲Every generation of CAD researchers dreams to be part of a generational shift to the high-level.

Today, it is called ESL

- Means many things
  - Algorithmic design and implementation
  - Behavioural synthesis
  - SoC construction, simulation and analysis
  - Virtual system prototyping
  - Function-architecture co-design
- Of course, it is (always) really about raising the level of abstraction for design…

The "KoolAid" about High Level

- Higher productivity
  - "designer productivity falls within 6 days/line to 6 lines/day regardless of the abstraction level"
  - Higher abstraction level means less coding
- Less bugs
  - "one bug per six lines regardless of the abstraction level"
- Improved design quality
  - Larger scope of design optimizations
- Shorter design time
  - Reuse of IP designs captured in executable specifications
- Indeed, several attempts to get this programming right
  - LISP, ADA, Prolog, Java, and many many variants of C and C++
- HLS has been a major preoccupation of the EDA community since late 1970s..

Sample Time Points (purely from recollection)

1978 McFarland: ValueTrace
1981 Kuck etc: Compiler Opt. POPL
1983 Hitchcock & Thomas: DP Syn.
1984 Gircyz thesis
1985 Kowalski & Thomas: AI
1986 Marwedel: Mimola
Orailoglu,Gajski: Flow Graphs
Parker: MAHA
Tseng & Siewiorek
1987 Trickey: Flamel
Ebcioglu: SW pipelining
1988 Brayton: Yorktown Silicon C.
Thomas: SAW
Ku & DeMicheli: HardwareC
Lam: SW pipelining / Lee: DSP

1989 Goosens, DeMan: loops
Paulin & Knight: FDS
Walker & Thomas
1990 Olympus
McFarland,
Parker, Camposano
DeMan: CATHEDRAL II
1991 Stok, Bergamaschi
Camposano & Wolf book
Hwang, Lee, Hsu: Sched.
1992 Gajski HLS book
Wolf: PUBSS
1994 DeMicheli Book
1996 Knapp Book

HLS Vision: Circa 1980s

- "From Behavior to Structure", "From Algorithm to Circuit"
- A very active community of researchers in "High Level Synthesis"

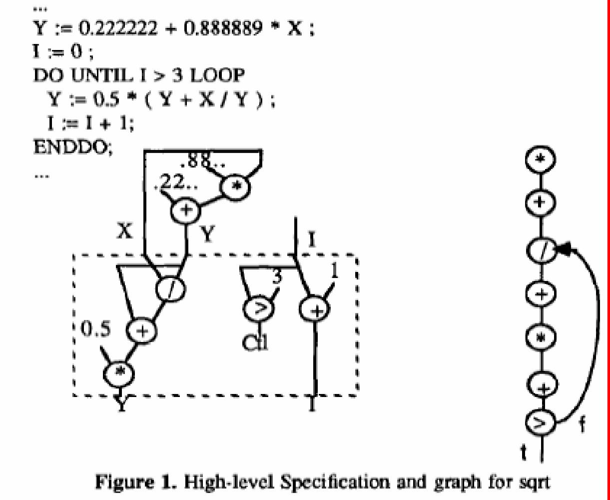


Figure 1. High-level Specification and graph for sqrt

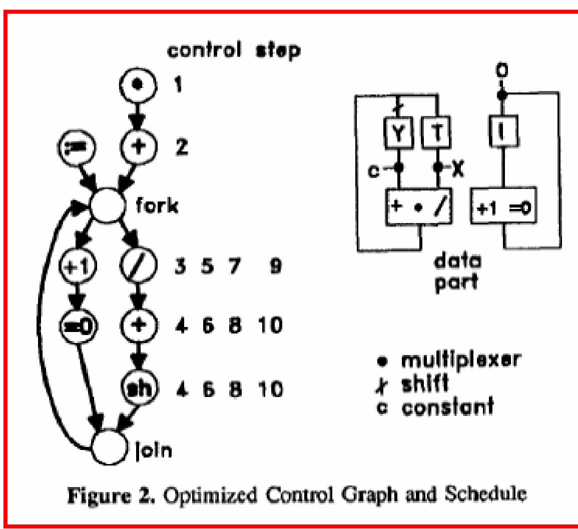


Figure 2. Optimized Control Graph and Schedule

- A compelling vision, neatly laid out problems, tasks
  - Then what happened?

➲**Answer: The dogs did not like the dogfood.**

Why? The Dogs and Their Food

- A partial answer…
  - Circuit designer's did not like the way to get to (known) results
    - And when they got there, the results were underwhelming
  - Shifts in design tools and methods do not happen alone..
  - People (must) change too..
    - Architects: deal with too many turning knobs
    - ASIC Implementers: understand and apply what is really important to optimize (and what is not)? Multiple clocks, rails, domains, …
  - ► Such shifts must be an enabler: <u>new people doing new things</u>.
- Gartner will tell you tell you that much
  - And then something about the quality of results (QoR).

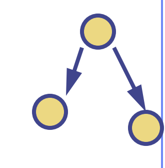
HL Modeling & Synthesis: A personal journey over 20 years

- My journey started as a circuit designer at Intel c. 1986
  - Life was 'Simple'
    - Simulation tool reproduced hardware behavior faithfully
    - Circuits hooked together: modularity & abstraction came naturally
    - DA for designers focused on methodological innovations (split runs, timing calculators, sanity checks)
    - Real simple handoff (of printed C-size sheets)
    - Local verifiability and updates through back annotations
  - Then things changed
    - Design became data, and data exploded
    - Programming paradigm percolated down to the RTL
    - Designers opened up to letting go of the clock boundary
- And we all asked:

Wouldn't it be fun to program the circuits?! At least, the dumb ones.

From HLL to HDL: Semantic Needs

MID 1980's



❶ Concurrency

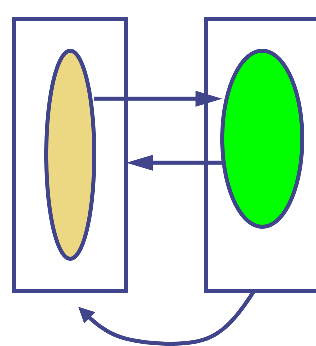
- model hardware parallelism, multiple clocks

EARLY 1990's

❷ Timing Determinism

- provide a "predictable" simulation behavior

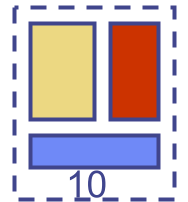
EARLY 2000's



❸ Reactive programming

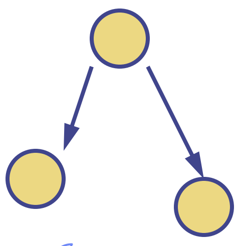
- provide mechanism to model non-terminating interaction with other components, watching, waiting, exceptions

MID 2000's



❹ Structural Abstraction

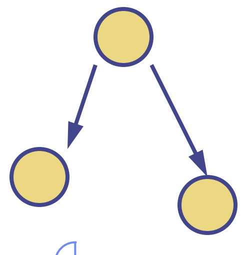
- provide a mechanism for building larger systems by composing smaller ones



Concurrency Experiments:
Example: HardwareC, Stanford circa 1989

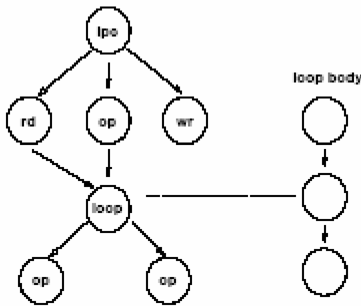
- Ambitious use of concurrency
  - Hierarchically nested blocks
    - [s1; s2] ; Sequential
    - { s1; s2 } ; Data-parallel
    - < s1; s2 > ; Force-parallel
- Focus on Scheduling smarts
  - Notions of bounded and un-bounded delay operations
- CDFGs ruled the day
  - Operational uncertainty captured in the structure of the model
- Memory was (often) an after-thought
  - Just another module

```
function memory-read(addr, data, ak, rq, val) return boolean[8]
    out port addr[16];      /* address line */
    inout port data[8];     /*  data  line  */
    out port ak;            /* request line */
    in port rq;             /* request line */
    in boolean val[16];     /* addr to read */
[
    while ( rq )          /* wait */
        ;
    <
        write ak = 1;            /*  take  line  */
        [
            write addr = val;    /* put address */
            return_value = read(data);
        ]
    >
    write ak = 0;
]
```



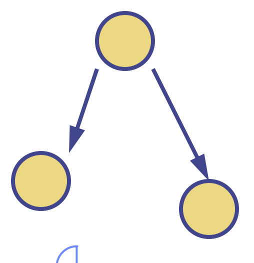
Lessons Learnt

- The Good
  - Not all CDFGs created equal
  - For instance: SIF
    - Match the model granularity to the problem solving methods
    - Structural handling of uncertainty
- The Bad
  - Too much concurrency is counter-productive
    - In fact, distinguish between concurrency and simultaneity
  - High control costs can not be avoided because of the model generality
- The Ugly
  - Picked the wrong door on language.



Timing uncertainty makes most concurrent programming languages a poor choice for modeling hardware systems.

-- IEEE D&T, November 1997



HLL to HDL: 3 ways to do it

One: Syntactic Add-on to match new concepts

- Process, Module, Signal, [], <>, channel, …

Two: Semantic overloads

- L_value = R_value implies…
  - E.g., an event into future

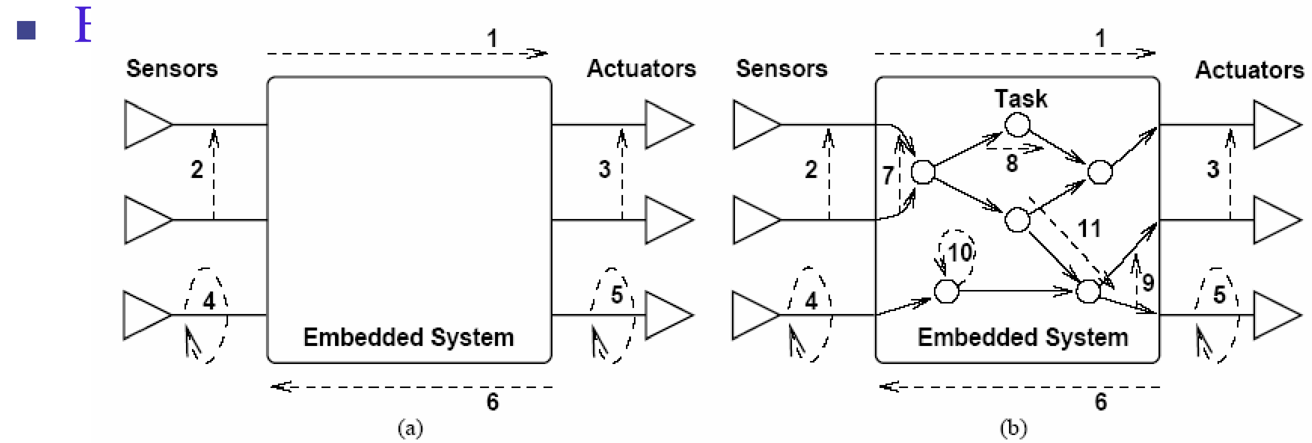
Three: Neither. Use existing mechanisms

- Libraries
- Operator overloading
- Polymorphism: port/type

Which would you choose?

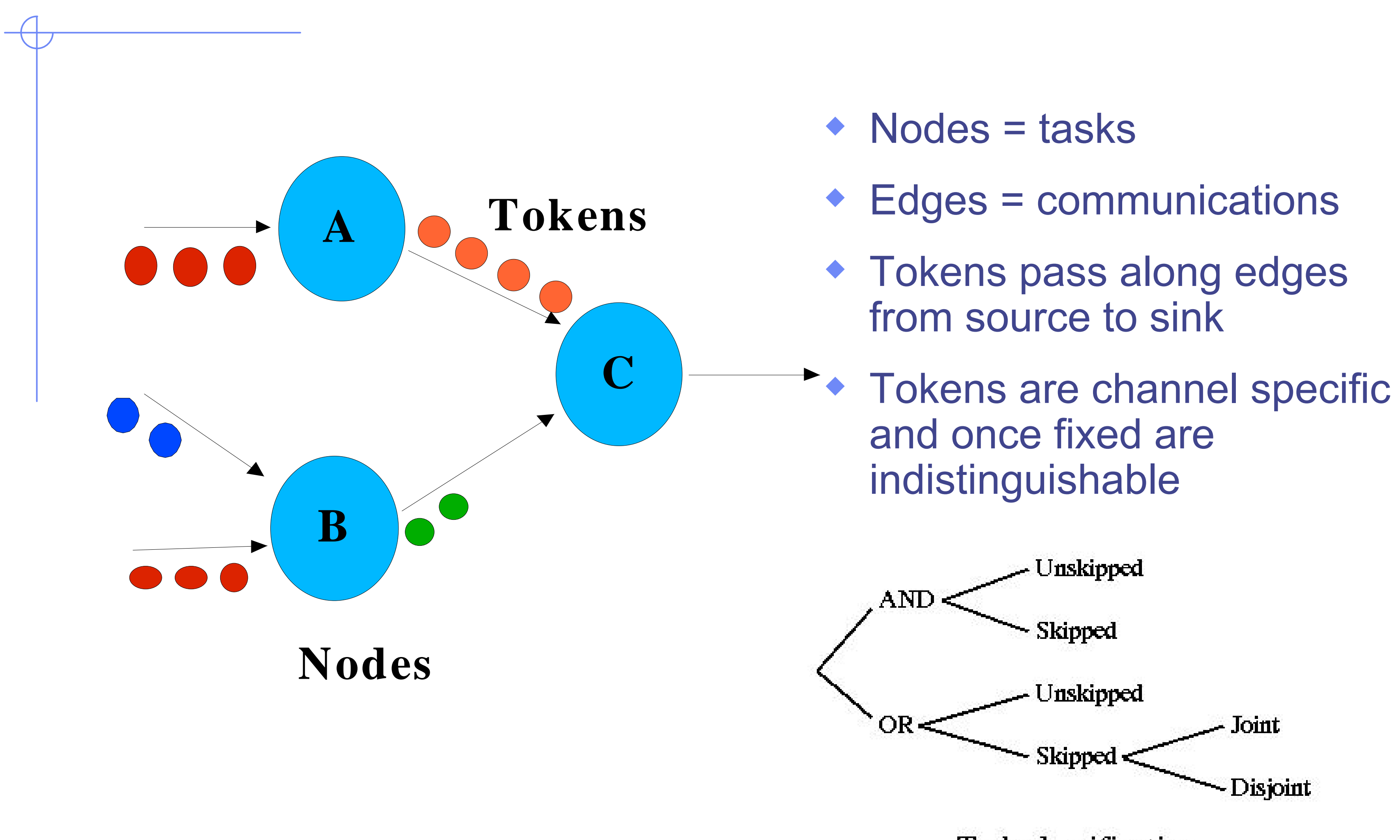
The Era Of Timing, Circa early 1990s

- Lexicon changed from the chip to the embedded system
- New ways of looking at the hardware (as an ES)
  - E



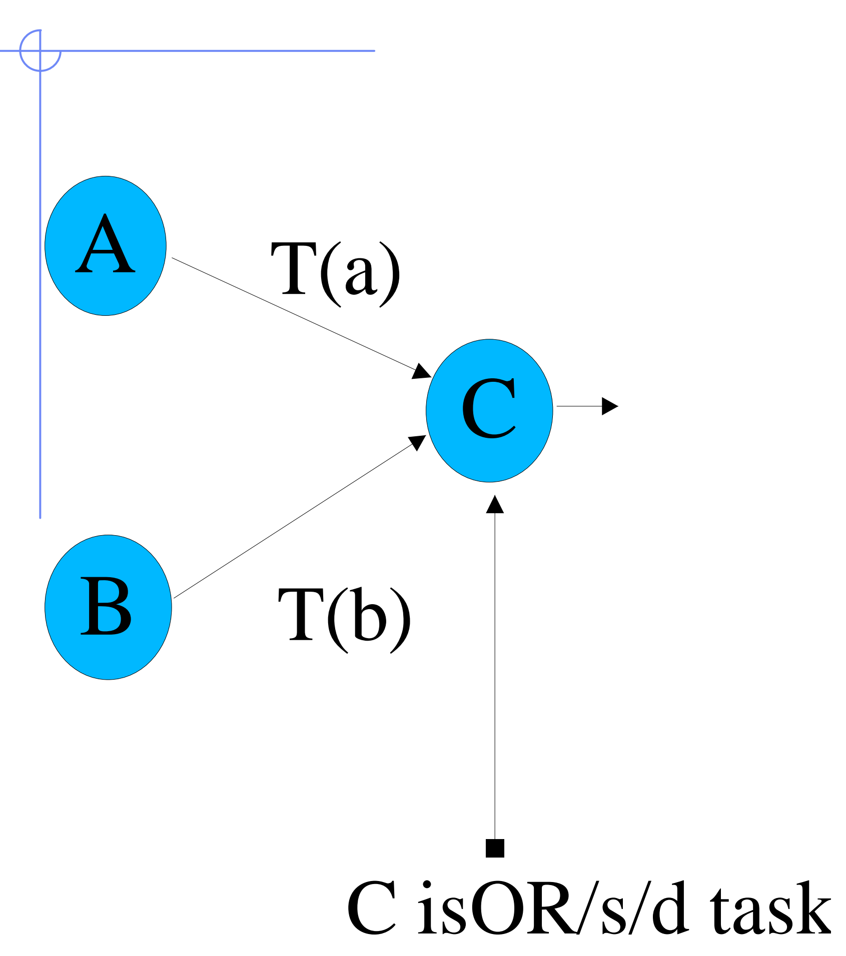
- By now, models did a full circle
  - From separate timing, function models to Operation-Event graphs to separate timing and task graphs.

Generalized Task Graph Model



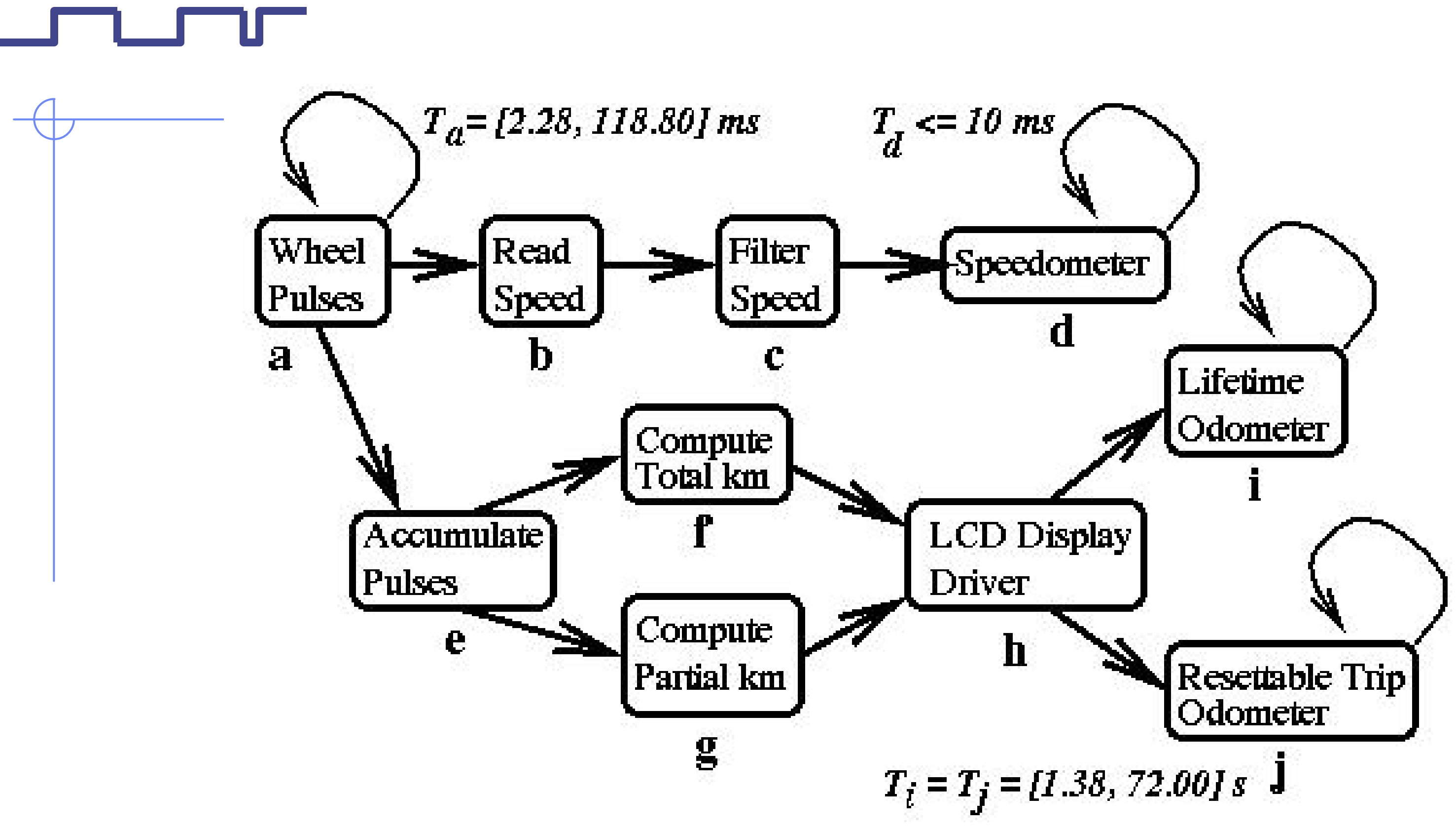
- Nodes = tasks
- Edges = communications
- Tokens pass along edges from source to sink
- Tokens are channel specific and once fixed are indistinguishable

Separation enabled 'Timing Simulation'




```
Always @(a or b) begin
    if (e != old_a) begin
        count_a = count_a + 1;
        mem[count_a] = a;
    end
... (similar change check for b)
    if (count_a >= T(a)) begin
        count_a = count_a - T(a);
        task_c(a, b);
    else if (count_b >= T(b)) begin
        count_b = count_b -T(b);
        task_c(a, b);
    end
end
```

Timing Simulation Example



Acceleration, deceleration periods:

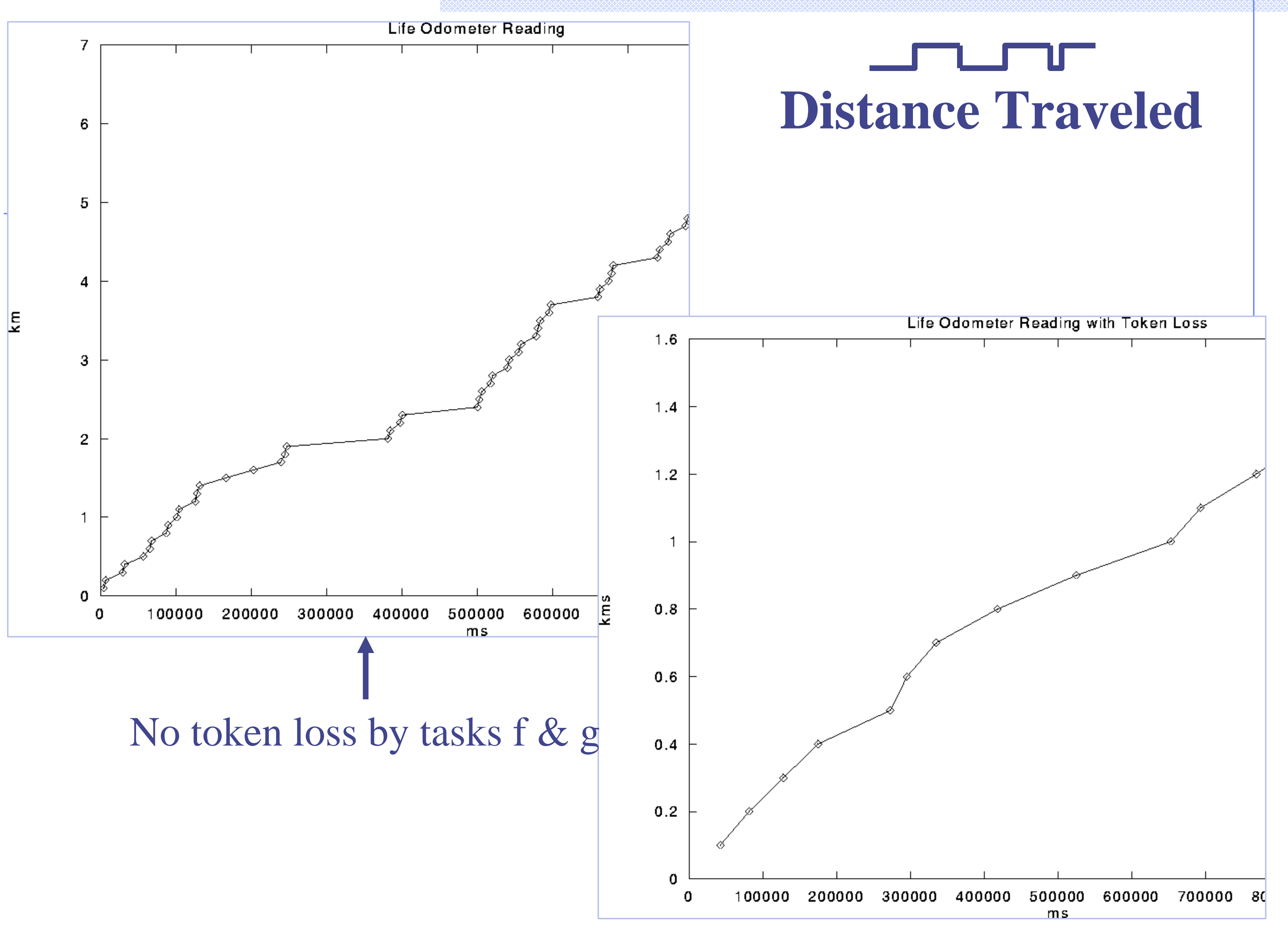
normally distributed with mean = 20 sec, dev. = 1 sec

Vehicular response:

normally distributed 10 sec/100 Kmph (10 +/- 4 sec)

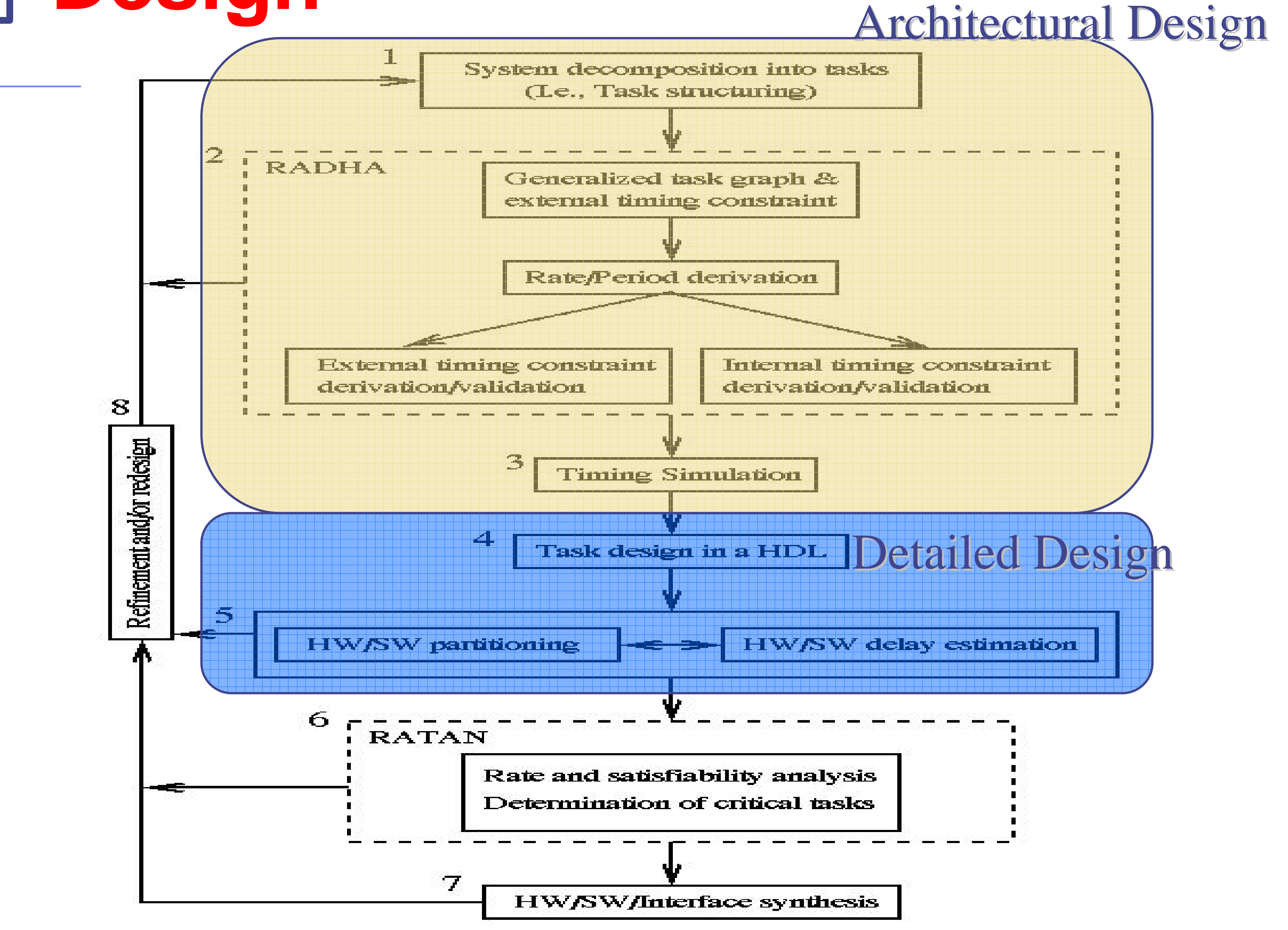
Hold speed for >= 2 x acceleration/deceleration period.

Distance Traveled



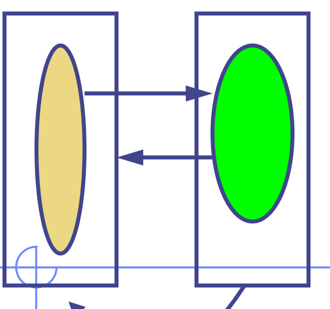
➲**System-level simulations *before* tasks have been implemented!**

Timing-Driven High-level Design



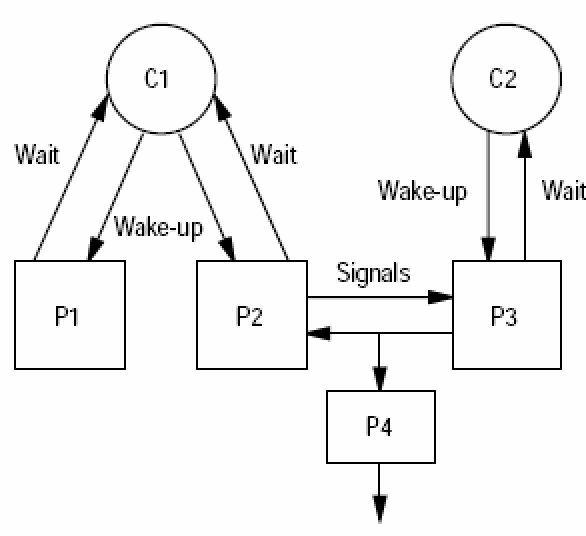
Lessons Learnt

- Too much, too little
  - A lot of detailed specification for solving only a part of the problem
    - Or handle an *even more* complex problem of time budgeting and constraint decomposition across modules
  - Especially, at a time when functional verification took on much increased importance.
- Model separation from function too limiting
  - And does not leverage the key capability of the designers to leverage function structure for timing
- The basic proposition in using HLL was lost
  - No chance of new formalisms and programming models to making timing first order.

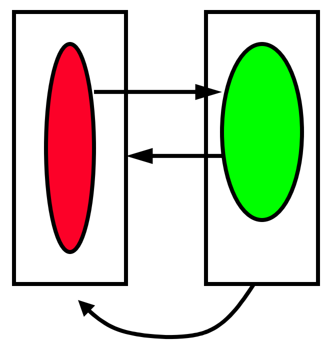


Reactive Programming: Mid 1990s, Scenic

- Inspired in part by the success of synchronous programming in embedded software
  - Esterel, Signal / Scade tools etc.
- Getting a better handle on "deterministic concurrency"
  - Early attempts to synthesize from Esterel
- Models crossed path with compilers & meta-models
- Enter Scenic/SystemC
  - Choice of the OO language
  - Reactivity: Watching versus Waiting
  - Libraries not syntax or overloading
- Marketed as iterative refinement on HLL programming



Modules, processes, reactions



Going from C++ to CSYN

Restricted C++ Description

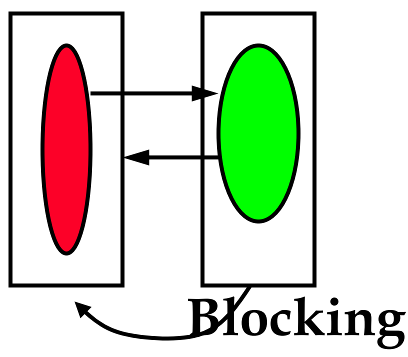
Add reactivity,
clock(s), waiting & watching

Refine data types
- bit true, fixed point
- saturation arithmetic

CONTROL

DATA

CSYN Description



Example: W & W

Blocking

```
csyn_signal<> a;
wait_until( a == '1');
block;
```

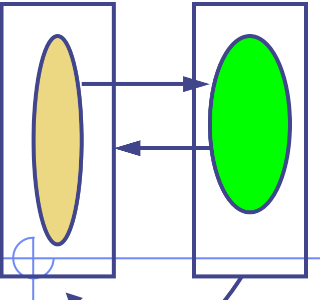
Non-blocking

```
csyn_signal<> a;
if (a.read() == '1') { }
block;
```

Con-current Watching

```
try {
normal_block
}
```

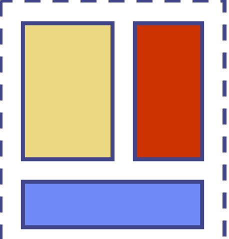
```
watching (a == '1');
catch (...) {
if (a.read() == '1')
{exeception_block }
```



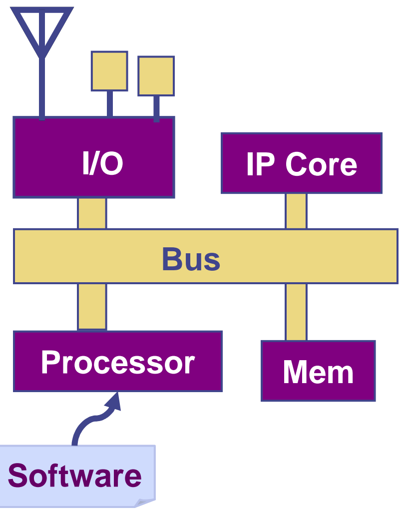
Scenic and UML

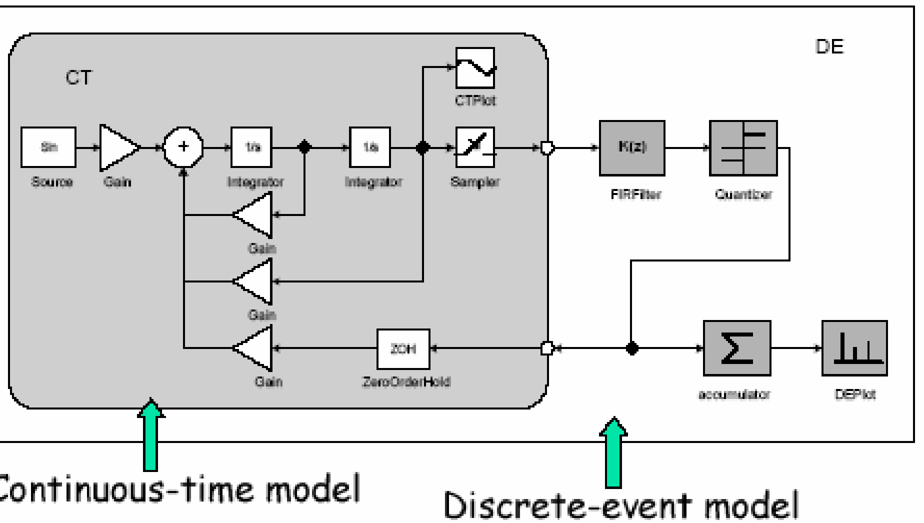
- Insight: expand model to include multiple *types* of relationships

1. Association:
   - unidirectional or bidirectional message passing
   - manifest themselves at run-time to permit exchange of messages among objects
   - associations are "structural," that is, they must be part of the class. Correspondingly objects have *links*.
   - implemented as pointers or references to objects.
2. Aggregation:
   - an object logically or physically contains another
   - physical or catalogue aggregation possible
     - often {shared} constraint used in two separate aggregations
   - may be recursive : may contain parts that may themselves contain classes of the original whole (although with different instances)
3. Composition:
   - aggregation plus owner is responsible for creation and destruction of the contained object
   - normally implemented as a pointer or reference, or declaration within the class scope
4. Inheritance: generalization or specialization
   - "is-a-kind-of relationship" that is fundamentally between classes (not invoked through messages)
   - the derived classes inherit properties from base class but may also extend or specialize them
   - "or-" or "and-" generalization
5. Refinement: generic or template elaborations



The era of structure: early 2000





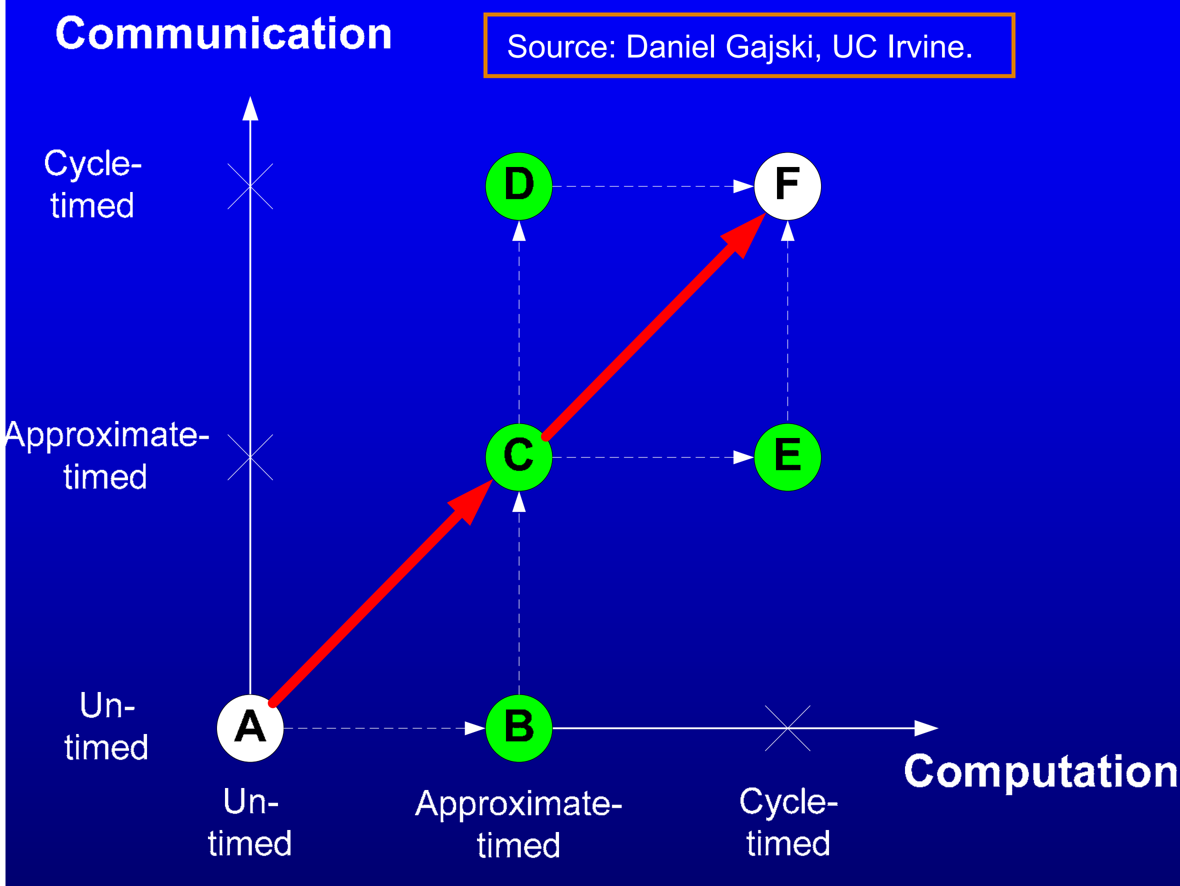
Modules,
Boxes,
Containers,
Wrappers,
IP,
Interfaces

MOCs, Meta Models, Process Algebra

Separation of Communications, TLMs

Component Composition Frameworks

Time Granularity in Models: Transactions



A. **"Specification model"**
"Untimed functioal models"

B. **"Component-assembly model"**
"Architecture model"
"Timed functonal model"

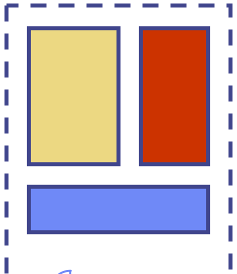
C. **"Bus-arbitration model"**
"Transaction model"

D. **"Bus-functional model"**
"Communicatin model"
"Behavior level model"

E. **"Cycle-accurate computation model"**

F. **"Implementation model"**
"Register transfer model"

- **Models B, C, D and E could be classified as TLMs**
  - Many many qualifications on TLMs: BCA/CA-TLM, Protocol aware TLM, SOC TLM, SOC-MA TLM, …



BALBOA Composition Framework

- A composition environment
  - Built upon existing class libraries, to add a software layer for manipulation and configuration of C++ IP models
  - Ease software module connectivity
  - Run-time environment structure
- A SW architecture that enables
  - composition of structural and functional information
- Current state
  - SystemC + NS2 + ISS + OS services

System designer

Component Integration, CIL

Compiled

Split-Level Interface/BIDL

C++, SystemC

Example

```
# instantiate components
Adder          a
Register       r
connect  a.z to r.in

# type introspection
a query type
⇒Adder

a query type parameters
⇒DATATYPE (bv10)

a query implementation
⇒add_fast<bv10>

a query ports
a b cin z cout

a.cin query type
bv<10>
```

CIL

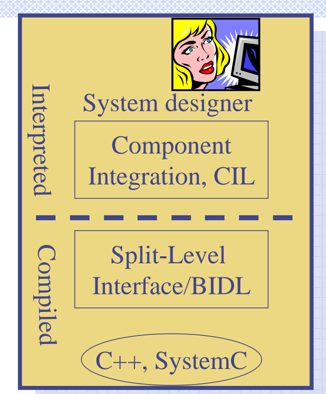
```
# Declare interface
Component Adder/interface {
  Inport  a
  Inport  b
  Inport  cin
  …
  Type parameter (DATATYPE)
}

# Declare implementation
Component Adder/Implementation {
  DATATYPE (bv10): add_fast<bv10>
…
}
```

BIDL

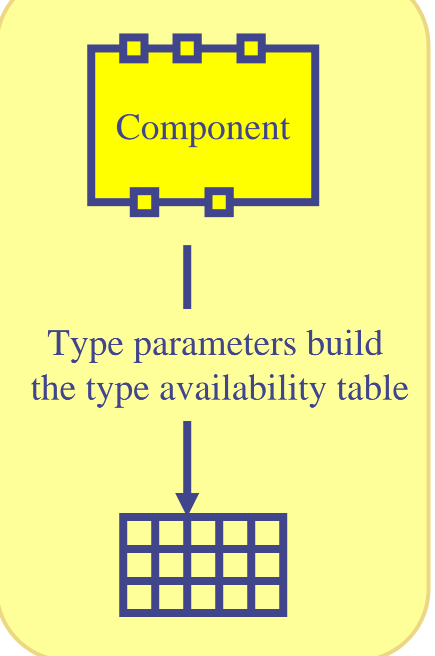
```
template<class T>
class add_fast: public sc_module {
  sc_in<bv10> a;
  …
};
```

C++

Type System in Balboa

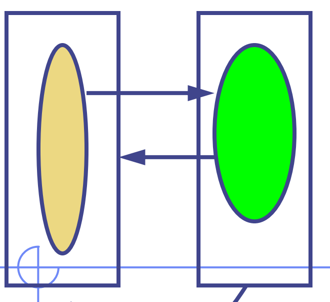


- Semi-lattice type relationship:
  - NP-hard to find a match for a netlist
    - Set P of ports partitioned into k sets (component)
    - Set S of signals
    - For each component, with its port vector p, assign a row from the TAT table such that if there is a signal set is compatible.
    - (One-in-Three Mono 3SAT can be reduced to Type Inference)
  - Full type resolution is not guaranteed
- Solved as a constrained optimization problem
  - If a component is not typed in the CIL
    - The SLI delays the instantiation of the compiled internal object
    - Interpreted parts of the component are accessible
  - Verify if types are compatible when a relationship is set
    - If a compatible type is found, the SLI allocates the internal object and sets the relationship
    - If not, the link command is delayed until the types are solved



In_1	In_2	Out_1	Out_2
float	float	float	bool
int8	int8	int8	bool
int16	int16	int16	bool
int32	int32	int32	bool
int64	int64	int64	bool
bool	bool	bool	bool

Reference: TCAD, Dec 2003



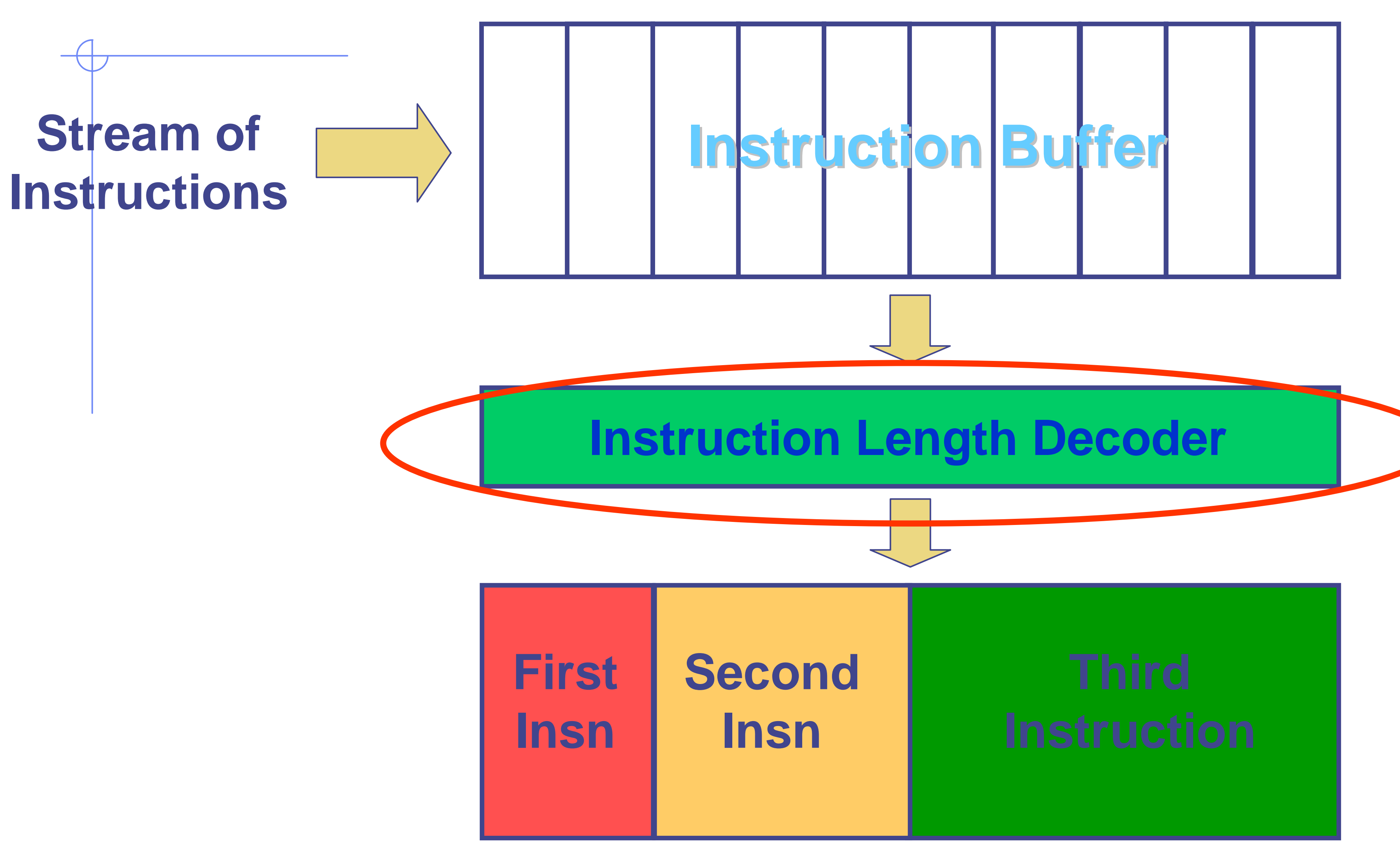
Lessons Learnt

- Can not sell new ways of doing the *same* thing to the same person who was doing it before
- Doing new things requires meaningful advance in new capabilities
  - E.g., where is support for verification, signoff?
- For a new group of people to pick of known methods, there must be a well defined target of methods and tools to retrofit
  - E.g., circuit design exploration by RTLers must bring circuit design into the RTL lexicon

Meanwhile…

- Companies found HLS underwhelming
  - At least, those with the $$ to buy tools
- Why? Was it QoR?
  - Nah…HLS did not address the real problems
    - E.g., Microprocessor functional blocks are typically
      - Low Latency: Single or Dual cycle implementation
      - Consist of several small computations
      - Intermix of control and data logic
- They wanted to start where HLS ended and go somewhere else
  - Start with a sequential, multi-cycle specification
  - Produce highly parallel, single-cycle design

Case Study: Intel Instruction Length Decoder



ILD Synthesis:

```
ResetArray(Mark);
NextStartByte = 0;
for (i=0; i < n; i++) {
  if (i == NextStartByte) {
    lc1 = LengthContribution_1(i);
    if (Need_2nd_Byte(i)) {
      lc2
      if (
        lc
        if
      }
    } el
      Length = lc1 + lc2;
  } else
    Length = lc1;
  } /* if (i == NextStartByte) */
  len[i] = Length;
  NextStartByte += len[i];
  Mark[i] = 1;
} /* end of for i loop */
```

Speculate Operations, Fully Unroll Loop, Eliminate Loop Index Variable

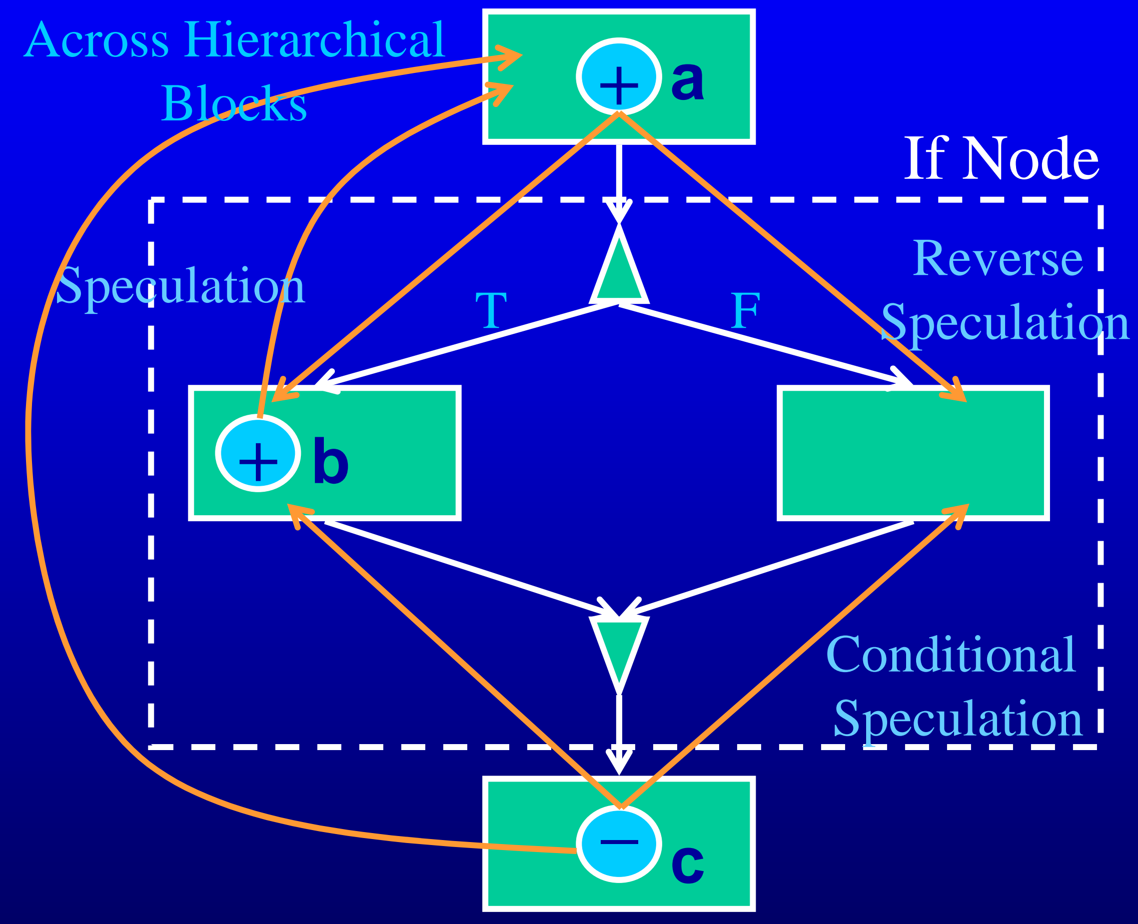
```
2,3);
3,4);
s(0));
s(1));
  NextStartByte += length[0];
  Mark[0] = 1;
}
if (1 == NextStartByte) {
  NextStartByte += length[1];
  Mark[1] = 1;
}
...
```

Multi-cycle Sequential Architecture → Single cycle Parallel Architecture

Another Attempt: Parallelizing HLS

- Vision
  - Aggressive & global code motions in an attempt to get past the QOR issue in HLS
- Strategy
  - Identify *really useful* parallelizing transformations
  - Apply coarse and fine grain HL & compiler optimizations
    - target control flow transformations
    - “Fine grain” loop optimization techniques for multiple and nested loops
    - Mixed IR suitable for fine and coarse grain compiler transformations (similar to other systems such as SUIF)
  - Make it accessible through C, SystemC

SPARK: Parallelizing Transformations



Operation Movement to reduce impact of Programming Style on Quality of HLS Results

Increasing the scope of Code Motions by Inserting New Scheduling Steps

Resource Constraints + −

If Node

BB 0

T F

BB 1

BB 2

a

b

c

d

BB 3

BB 4

e

If Node

BB 0

T F

BB 1

BB 2

a c

b

d

BB 3

BB 4

e

S0

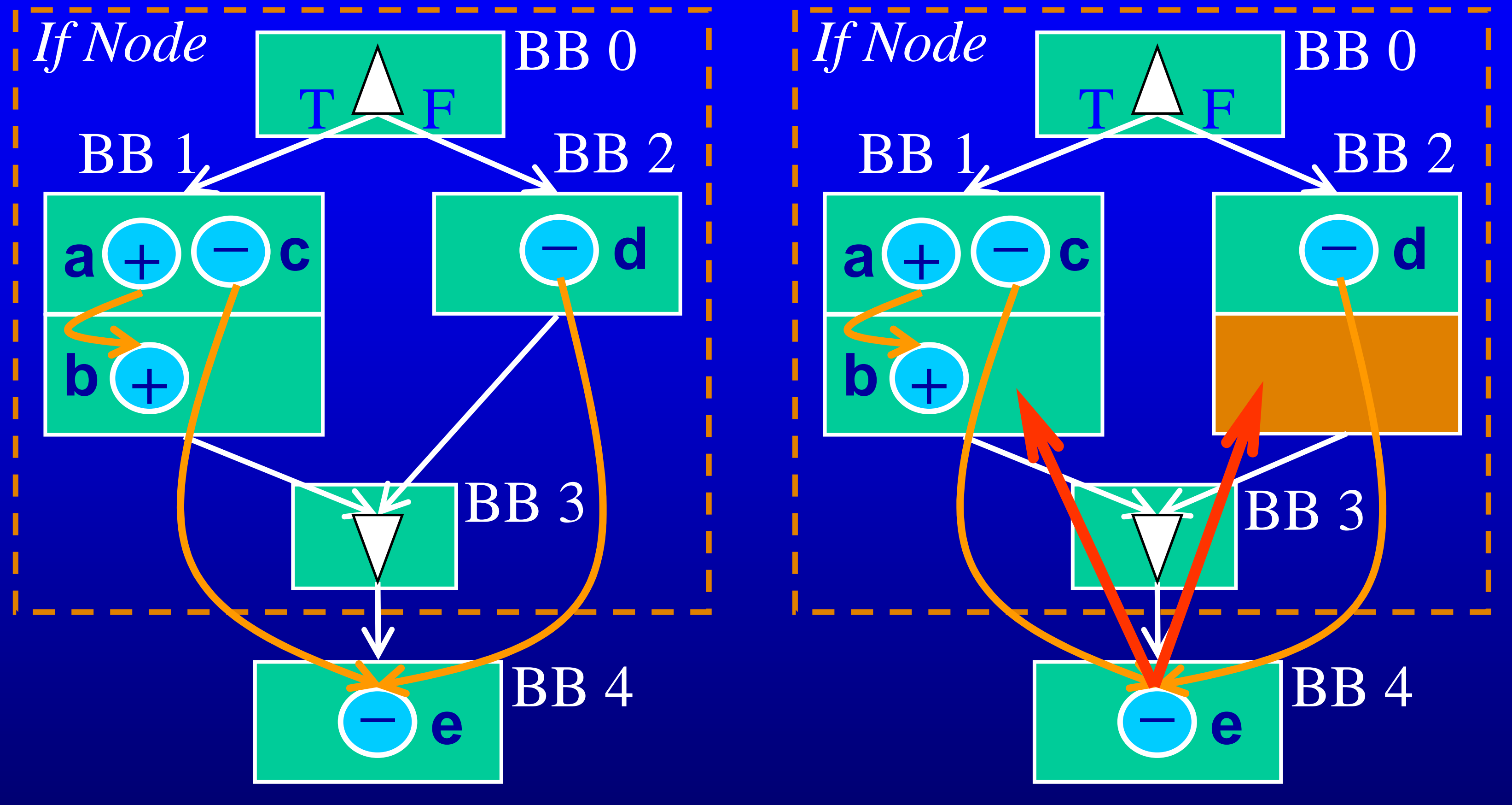
S1

S2

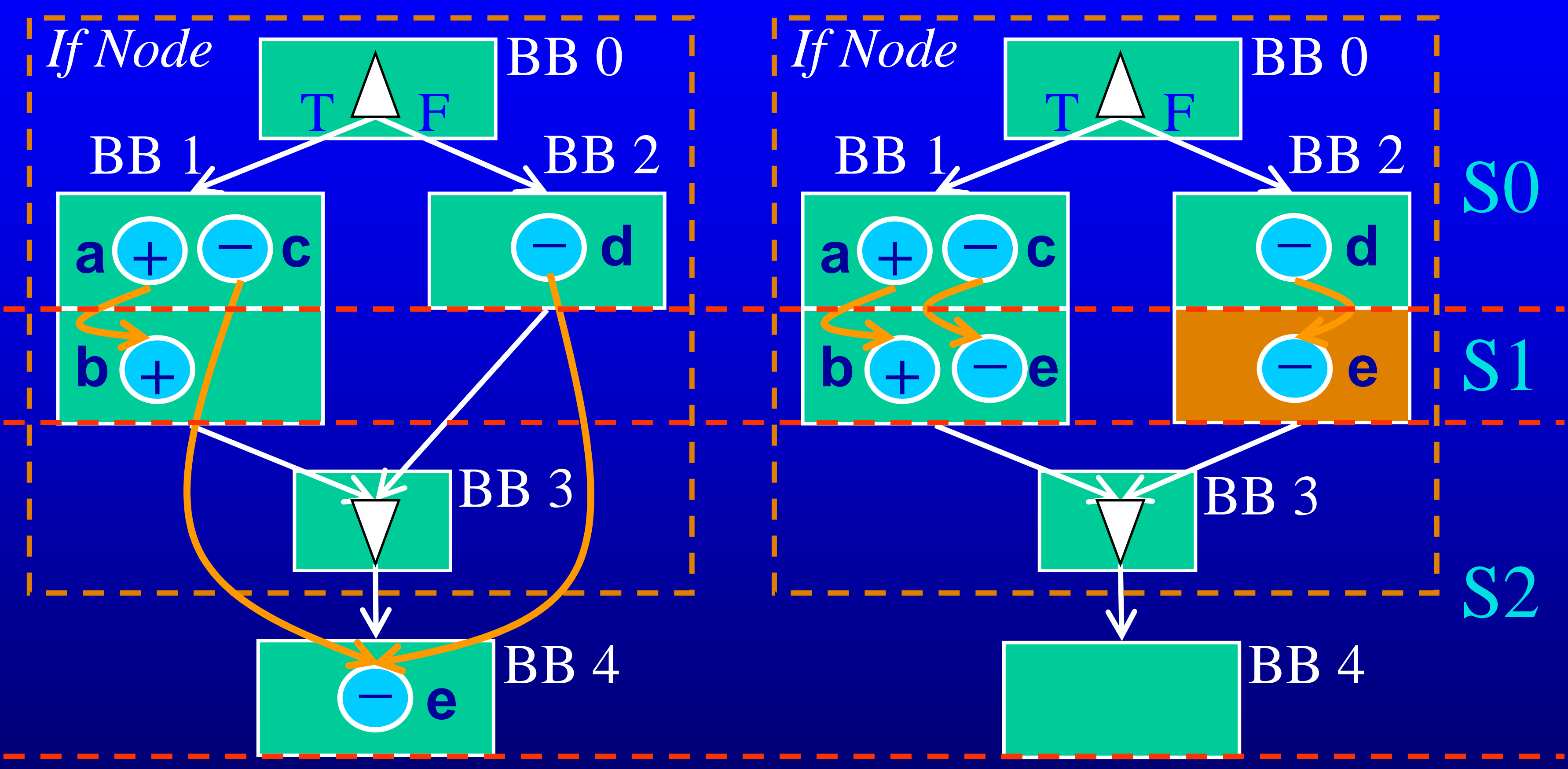
S3

Unbalanced Conditional

Inserting New Scheduling Steps



Enables Conditional Speculation

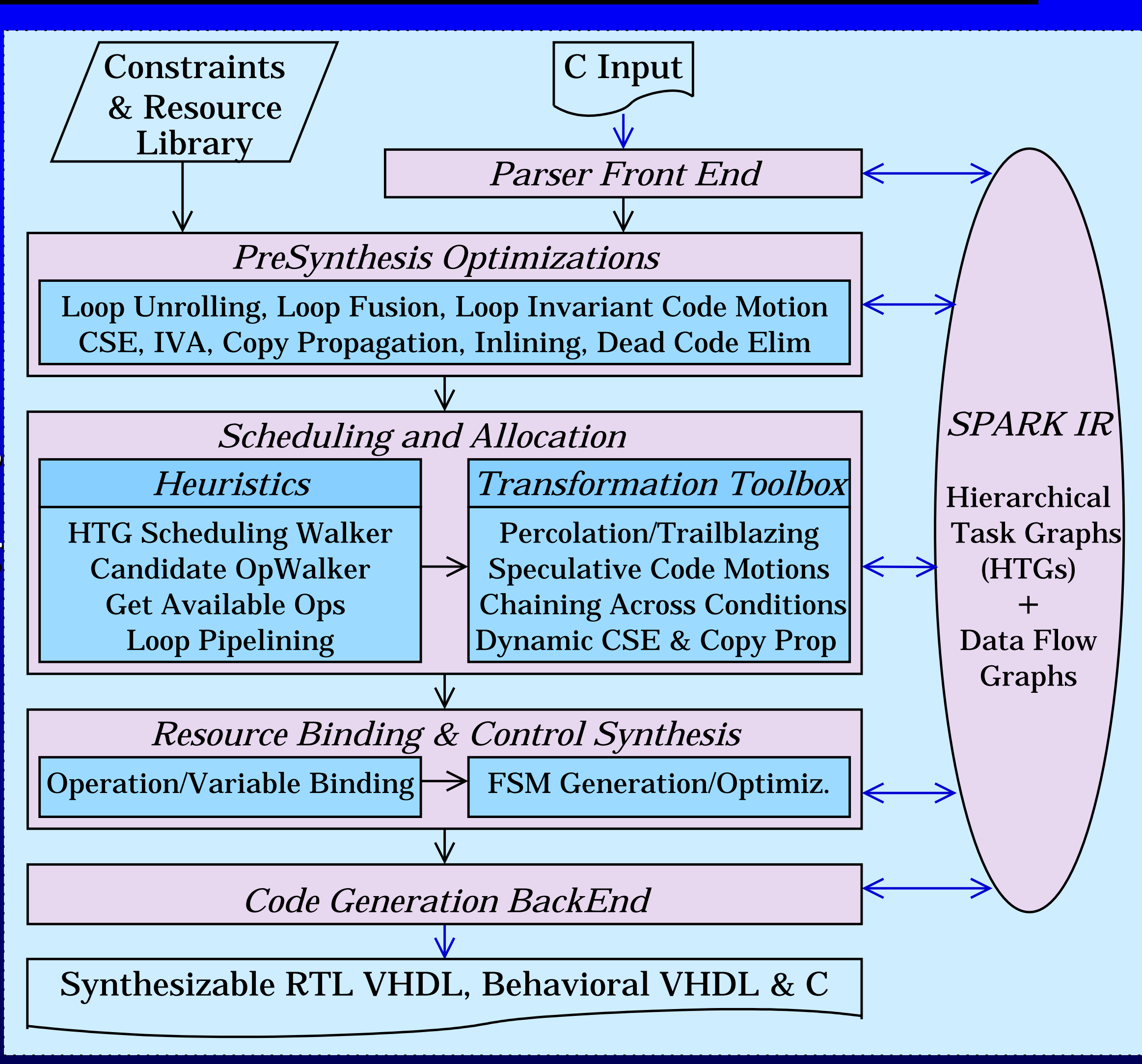


- Insert scheduling steps into shorter conditional branch
- Enables further code compaction

SPARK

Transformation Groups:

- **Pre-synthesis:**
  - Loop-invariant code motions, Loop unrolling, CSE
- **Scheduling:**
  - Speculative Code Motions, Multi-cycling, Operation Chaining, Loop Pipelining
- **Transformations applied dynamically during scheduling:**
  - Dynamic CSE, Dynamic Copy Propagation
- **Basic Compiler Transformations:**
  - Copy Propagation, Dead Code Elimination



HTTP://MESL.UCSD.EDU/SPARK

So, that brings us here to 2006

- What have we learnt and how do we go forward?

TakeAways

#1 HLS must be an enabler for the designer

- Who are we enabling? System Architects? COTS programmers? Mathematicians?
- The needs are real:
  - Architects have to deal with too many turning knobs
  - ASIC Implementers: understand and apply what is really important to optimize (and what is not)? Multiple clocks, rails, domains, …

#2 Address pain points of the identified target

#3 Move in step with the technical needs

- HLS was way ahead of the validation curve, even before the designer was ready to yield the clock boundary

#4 We need to bring the excitement back into the domain

- System design is mired in a lot of ‘black art’. Go from art to science: e.g., new methods to capture and exploit meta-data.

Address HL Pain Points

PP1: Components and Compositions

- Compositional models and methods for IP

PP2: Correctness, Security

- Ensuring and demonstrating correctness, confidence in design

PP3: Low Power and Power Management

- This one is at all levels

PP4: Flexibility, Programmability and Programming

- Improve silicon efficiency

PP5: Effective integration with BEOL

- Be closer to the project execution paint points.

PP3: Power, where is the pain most keenly felt?

- Making Architectural design with power specific decisions
  - What events do I wake up on and what events to use for scaling v/f?
  - Policies to move gradually and correctly among power/performance states
- Dilemma in binding and allocation of V/F ranges to blocks
  - Late binding of parameters to technology-specific values
  - Yet early determination of control of these parameters: architecture, sw
- Deciding μarchitectural choices in power gating
  - E.g., whether the state information is saved explicitly at the microarchitectural level or whether circuit strategies are used, such as retention flops on a backup power grid.
- Decisions with different area, power, performance/energy tradeoffs.
  - Complicated μarchitectural design, mode switching, pipeline design. Need estimates on energy savings.
- Verify if the power state controller is working or not
  - Ensure functional behavior, ensure compliance (standard-specific power related behavior, e.g., wakeup interval bounds)
- Re-evaluate and/or validate power state decisions at the gate level

Formal Performance Verification

From PCI Express:

PMG.02.00#10: After successful completion of the L2/L3 ready transition protocol a Link must transition to L3 when main power is removed if the system does not provide a Vaux supply. It must not transition before the main power is removed.

PMG.02.00#02: All power supplies, component reference clocks, and component's internal PLLs must be active during L0 and L0s.

PMG.02.00#06: All platform provided power supplies and component reference clocks must remain active during L1.

Power Management Checklist

ID	Requirement	Answer
PMG.01.01#01	Root complexes are required to participate in Link power management DLLP protocols initiated by the downstream device.	Yes __ No __
PMG.01.01#02	Active State Link Power Management using the L0s state must be supported by all PCI Express components.	Yes __ No __
PMG.02.00#01	All PCI Express components must support the L0 active state.	Yes __ No __
PMG.02.00#02	All power supplies, component reference clocks, and component's internal PLLs must be active during L0 and L0s.	Yes __ No __
PMG.02.00#03	No TLP or DLLP communication is allowed over a side of a link in the L0s state.	Yes __ No __
PMG.02.00#04	No TLP or DLLP communication is allowed over a link the L1 state.	Yes __ No __
PMG.02.00#06	All platform provided power supplies and component reference clocks must remain active during L1.	Yes __ No __
PMG.02.00#08	The L2/L3 Ready transition protocol must be supported.	Yes __ No __
PMG.02.00#09	TLP and DLLP communication over a Link that is in L2/L3 Ready is prohibited.	Yes __ No __
PMG.02.00#10	After successful completion of the L2/L3 ready transition protocol a Link must transition to L3 when main power is removed if the system does not provide a Vaux supply. It must not transition before the main power is removed.	Yes __ No __
PMG.02.00#11	An upstream initiated transaction targeting a Link in L0sor L1 must cause the Link to transition back to L0.	Yes __ No __
PMG.02.00#15	TLLP and DLLP communication over a Link that is in L2 is prohibited.	Yes __ No __
PMG.02.00#16	TLLP and DLLP communication over a Link that is in L3 is prohibited.	Yes __ No __
PMG.02.00#19	A component may only consume VAUX power if enabled to do so.	Yes __ No __
PMG.03.08#14	When an upstream component receives PM_ENTER_L23_DLLP it must reply with the PM_Req_ACK DLLP.	Yes __ No __
PMG.03.09#04	Upon receiving a PM_Enter_L1_DLLP an upstream component must complete all outstanding TLPs and block scheduling of new TLPs.	Yes __ No __
PMG.03.09#05	The upstream component that received a PM_Enter_L1_DLLP must send a PM_Request_Ack_DLLP downstream once all its outstanding TLPs have completed and it has accumulated at least the minimum number of credits required to send the largest possible packet for any FC type. It must send this DLLP continuously until it receives an electrical idle set or observes its receive lanes enter the idle state.	Yes __ No __
PMG.03.09#07	When an upstream component observes its receive lanes enter the electrical idle state it must stop sending PM_Request_Ack DLLPs, and disable its Link layer, send one electrical idle ordered set and bring its transmit Lanes to electrical idle.	Yes __ No __
PMG.03.09#09	Once both ends of a link are in the L1 state the upstream component must suspend operation of Flow Control Updates.	Yes __ No __
PMG.03.10#01	An upstream component must detect when a packet is targeted at a downstream Link in the L1 state and initiate the transition of the link to L0	Yes __ No __

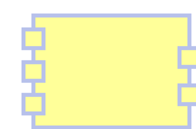
Page 1 of 3
Almost the same size as System Architecture Checklist.

3x the electrical requirements.

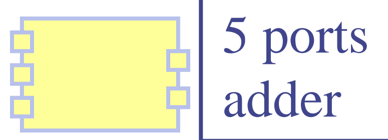
Can not all be done by FV tools that do not have a good handle on voltage levels, on/off domains, back biasing, …

Reflection and Introspection: A HW Guy's Way of Looking At It

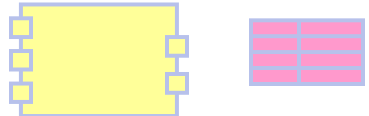
- Component:
  - A unit of re-use with an interface and an implementation



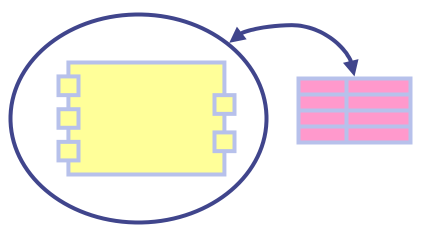
- Meta-information:
  - Information about the structure and characteristics of an object



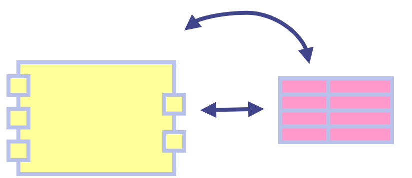
- Reification:
  - A data structure to capture the meta-information about the structure and the properties of the program



- Reflection:
  - An architectural technique to allow a component to provide the meta- information to himself



- Introspection:
  - The capability to query and modify the reified structures by a component itself or by the environment



Emerging 'meta data methods'

- Internet programming has many shared needs
  - Programming with data/methods from diverse sources, semi-structured data, platform independence, lightweight
- Focused on "data" (not document) transfers through XML schemas
  - Self-documenting/extensible "tags", extended through nesting
- In graph representation, there is no distinction between data and schema
  - Simplest XML is a labeled ordered tree with labels on nodes, and possible data values at the leaves.
- Schema extracted through Data Type Definitions
  - A DTD is an extended CFG with no terminals:
    - Nonterminals are tags in the XML parse tree
    - A document satisfies a DTD if it is a derivation of the ex
  - Not quite a data type in a programming language:
    - Values are not constrained (all values as strings);
    - Unordered things are difficult;
    - Inability to separate type of an element from its name.
- New flexible types and schemas, e.g., regular expressions over trees => Ability to talk "about" data / queries through reflection

<dealer> <UsedCars> <ad>
<model>Honda</model><yr>92</yr>
</ad></UsedCars>
<NewCars> <ad>
<model>Prius</model></ad></NewCars
> </dealer>

root: dealer
dealer → UsedCars, NewCars
UsedCars → ad*
NewCars → ad*
ad → modelyear | model

Summary

- The current movement towards HLM through programming advances holds the promise of modeling and methodology convergence from chip design to embedded systems (software) design
  - Language-level modeling advances now touching new compositional abilities through innovations in design patterns and infrastructure capabilities
- However, such advances go hand-in-hand with advances in verification and synthesis tools
  - Yet, good IP-model composability still very much out of reach
- New models and methods needed to
  - Capture design, design constraints, meta-information
  - To validate compositions, to drive design tasks that utilize meta-information.
  - To address power, reliability related questions at level where they can have most impact on the system architecture.