# Scaling Commercial Verification to Larger Systems

Robert Kurshan
Haifa
October 24, 2007

# Design Costs

- synthesis/layout < 50% total cost
  - more or less linear in chip size
- debug/verification = 50% - 80% of total cost
  - embedded software
  - n parallel components of size m leads to m^n system states
  - so functional verification grows exponentially with design size
- widely held that the cost of fixing a bug grows exponentially with the development stage at which it is detected/fixed
  - on account of increasing interactions with other components that also must reflect changes from fixes
- holding down costs leads to less test coverage and lower design reliability
  - but cost of product failures can also be high
  - ready for a $500M recall?

# The BIG Verification Problem

Verification (intrinsically) DOESN'T SCALE

– Component interactions grow exponentially with the number of system components, while conventional system test at best can increase coverage as a linear function of allotted test time.

– Likewise, capacity limitations are commonly cited as the essential gating factor that restricts the application of automatic formal verification (model checking) to at most a few design blocks.

# How Can We Hold Design Costs In Check?

- Sacrifice design reliability
  - lower test coverage

- Limit design size

**OR . . .**

# The BIG Solution: *ABSTRACTION*

Abstraction has long been used successfully in pilot projects to apply model checking to entire systems. Abstraction in conjunction with guided-random simulation can be used in the same way to increase coverage for conventional test.

# Abstraction as Divide-and-Conquer

- Divide-and-conquer requires the precision of formal methods

- Types of divide-and-conquer
  - Horizontal (flat) decomposition – abstracts component environment
  - Vertical (hierarchical) decomposition – abstracts lower-level details

- Conservative vertical abstractions support verify-only-once: at highest level of abstraction where property is defined
  - Contrast with Transaction-Level Modeling

- Enables earlier debug
  - main power and innovation will come from vertical decomposition

# Vertical (Hierarchical) Decomposition

- Design development today: **data before control**
  - Controllers need to point to defined data structures
  - But: upside down – often need to modify data structures for controllers
- Decompose vertically: **control before data**
  - Use stubs as place-holders for data
  - Controllers point to stubs
  - Stubs are oracles for data path computation
- Imposes hierarchical decomposition
  - Control at higher levels (coarse granularity supports global verification)
  - Data paths at lower levels (fine granularity verified locally)
  - Constant complexity at each level – scales with increasing design size

# Vertical (Hierarchical) Decomposition, cont.

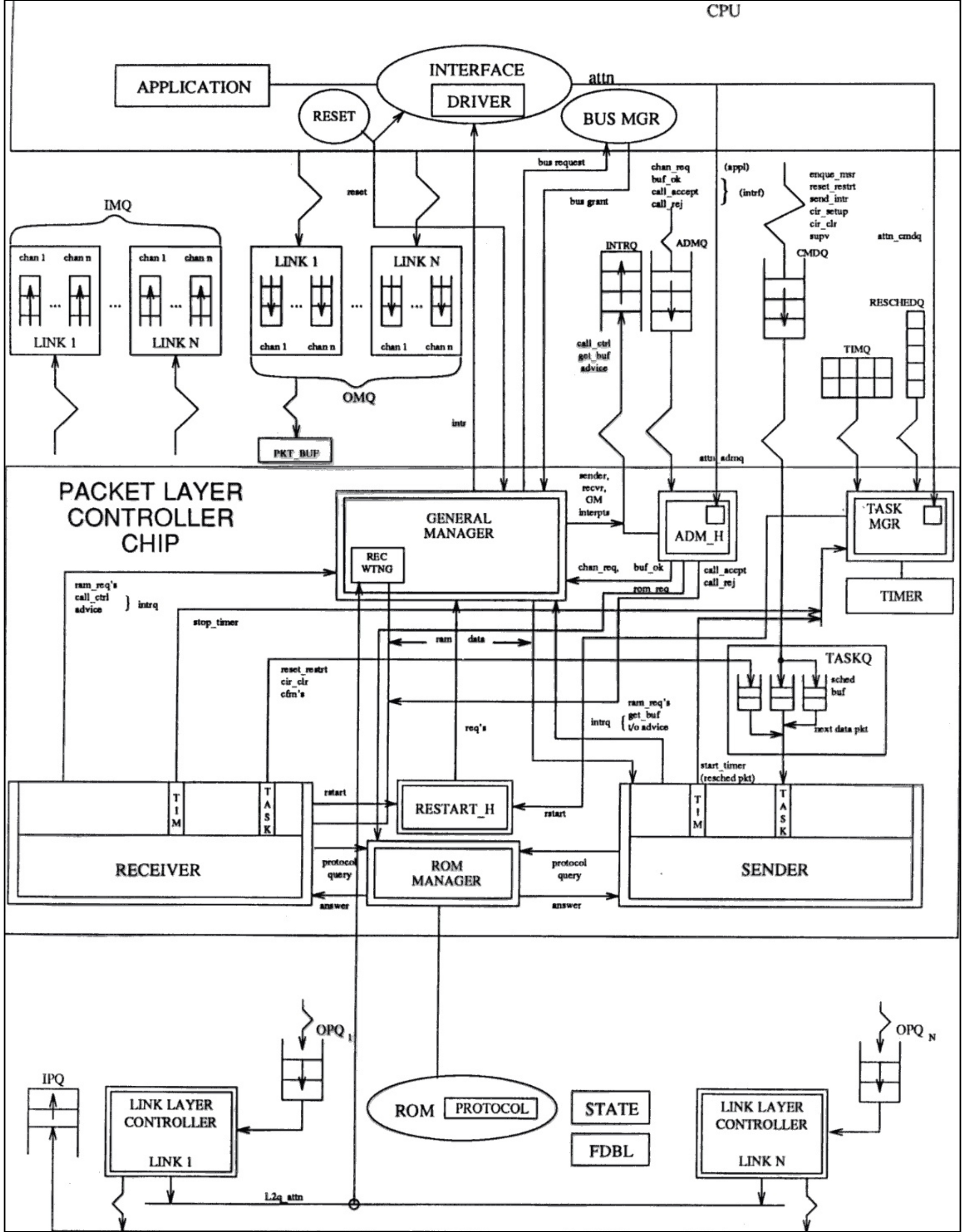
1. Start with functional spec, floor plan, etc
2. Derive properties (test plan) BEFORE coding design!
    1. Formal spec with comments
    2. Specification reviews (like design reviews) for completeness
3. Partition properties into levels
    1. Control properties first (global properties)
    2. Data path properties last (local properties)
4. Code to properties
    1. Use stubs as place-holders/oracles for lower levels
    2. Verify (simulation or formal) as you design

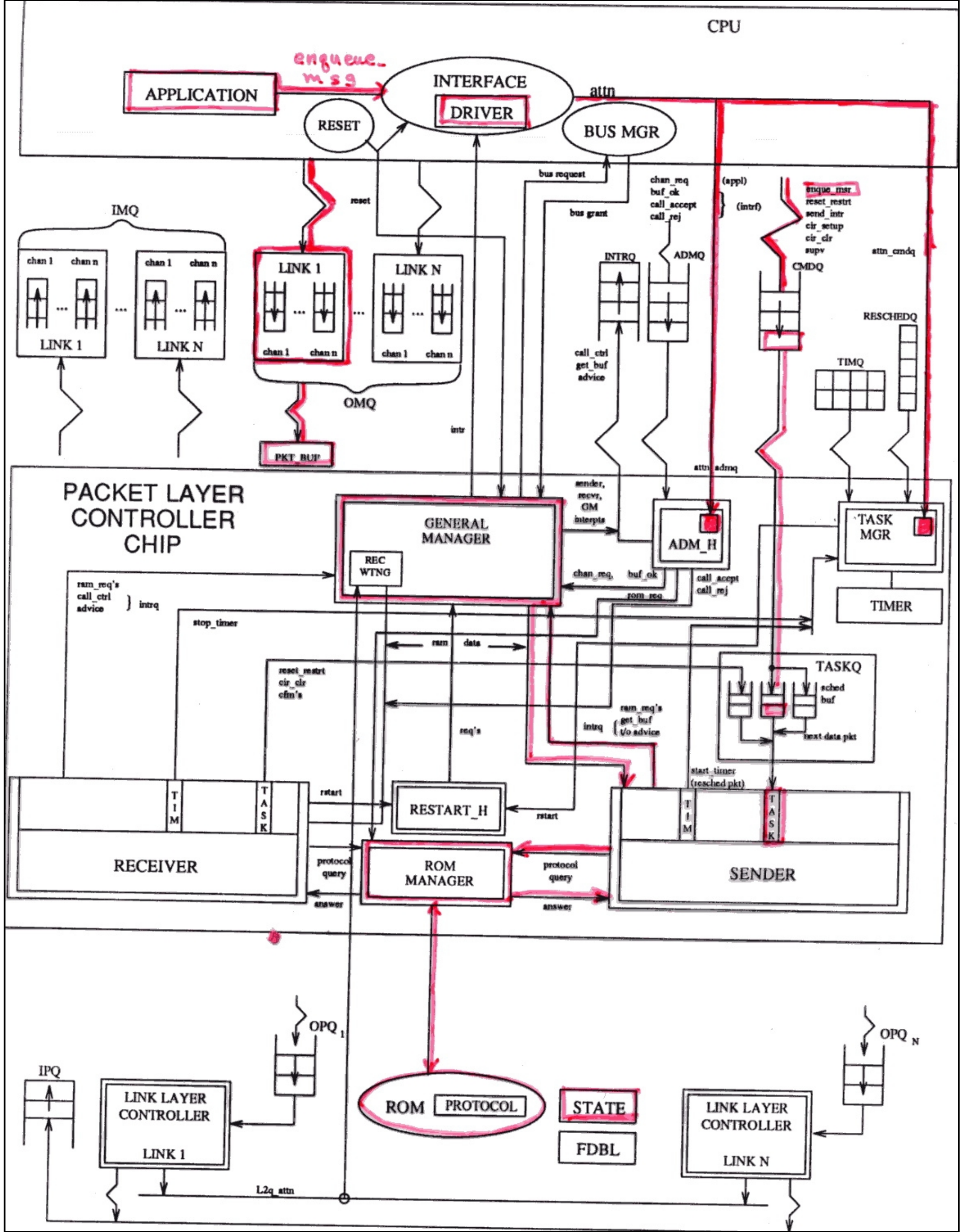
**Implements top/down – bottom/up hierarchical design process**

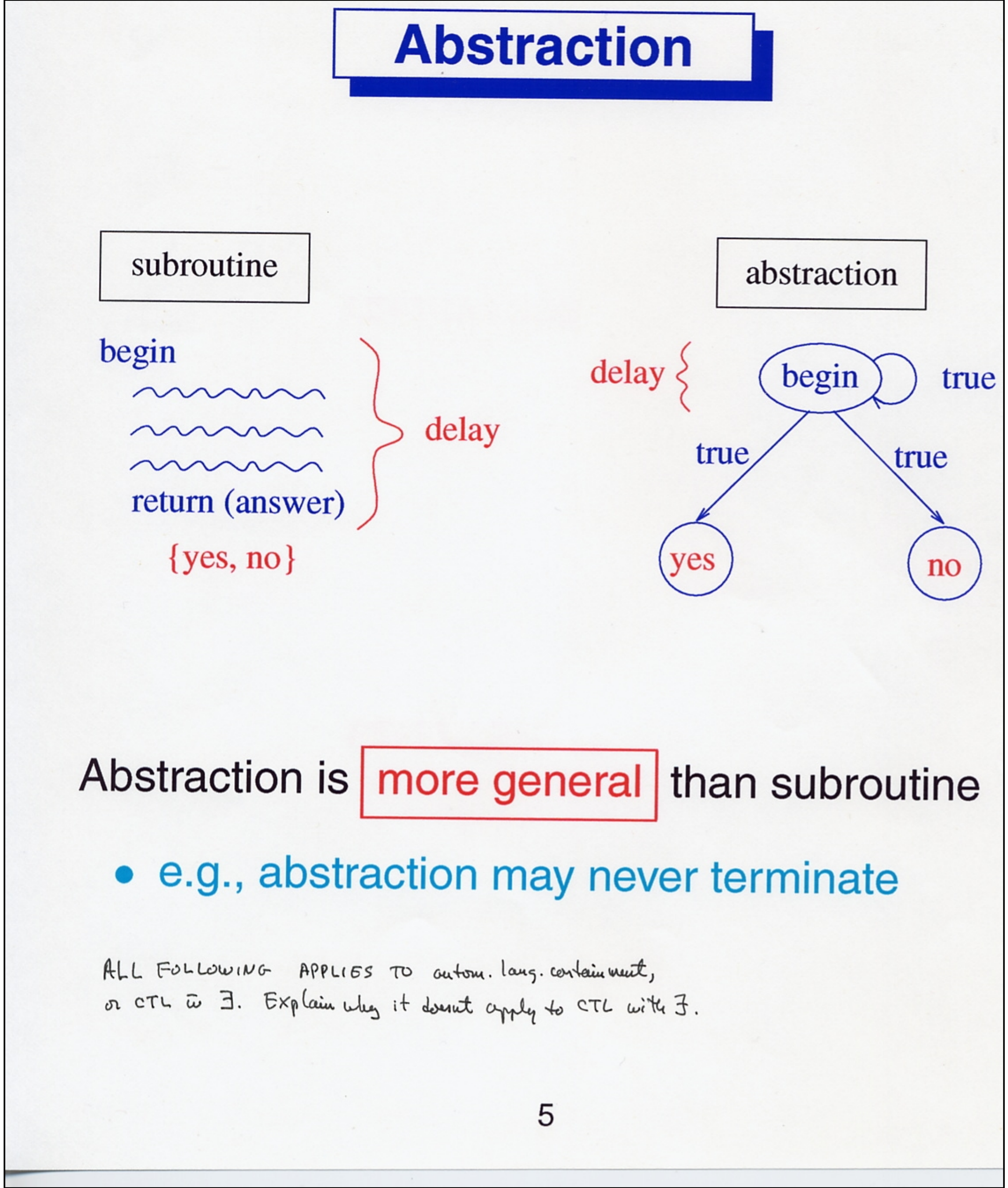
# 20 years ago …

## Packet Layer Controller chip development at Bell Labs

- 200,000 transistors
- Developed entirely under the control of formal verification through a top/down stepwise refinement hierarchy
- 20% of projected cost
  - 6 staff years/2 calendar years vs projected 30 staff years
- “reliability of a 2nd generic release”


CPU
APPLICATION
INTERFACE
DRIVER
attn
RESET
BUS MGR
bus request
bus grant
reset
chan_req
buf_ok
call_accept
call_rej
(appl)
(intrf)
enque_msr
reset_restrt
send_intr
cir_setup
cir_clr
supv
attn_cmdq
IMQ
chan 1
chan n
LINK 1
LINK N
OMQ
PKT_BUF
INTRQ
ADMQ
CMDQ
RESCHEDQ
TIMQ
call_ctrl
get_buf
advice
intr
attn_admq
PACKET LAYER CONTROLLER CHIP
GENERAL MANAGER
REC WTNG
sender, recvr, GM interpts
ADM_H
TASK MGR
TIMER
chan_req,
buf_ok
rom_req
call_accept
call_rej
ram_req's
call_ctrl
advice
intrq
stop_timer
ram
data
reset_restrt
cir_clr
cfm's
TASKQ
sched
buf
next data pkt
ram_req's
get_buf
t/o advice
intrq
req's
start_timer
(resched pkt)
T I M
T A S K
rstart
RESTART_H
RECEIVER
SENDER
protocol query
answer
ROM MANAGER
IPQ
OPQ 1
OPQ N
LINK LAYER CONTROLLER
LINK 1
LINK N
ROM
PROTOCOL
STATE
FDBL
L2q_attn


CPU
APPLICATION
enqueue_msg
INTERFACE
DRIVER
RESET
attn
BUS MGR
reset
bus request
bus grant
chan_req
buf_ok
call_accept
call_rej
(appl)
(intrf)
enque_msr
reset_restrt
send_intr
cir_setup
cir_clr
supv
attn_cmdq
IMQ
chan 1
chan n
LINK 1
LINK N
OMQ
PKT_BUF
INTRQ
ADMQ
CMDQ
RESCHEDQ
TIMQ
call_ctrl
get_buf
advice
intr
attn_admq
PACKET LAYER
CONTROLLER
CHIP
GENERAL
MANAGER
REC
WTNG
sender,
recvr,
GM
interpts
ADM_H
TASK
MGR
TIMER
chan_req,
buf_ok
rom_req
call_acept
call_rej
ram_req's
call_ctrl
advice
intrq
stop_timer
ram
data
reset_restrt
cir_clr
cfm's
TASKQ
sched
buf
next data pkt
ram_req's
get_buf
t/o advice
intrq
req's
start_timer
(resched pkt)
rstart
RESTART_H
rstart
T
I
M
T
A
S
K
RECEIVER
SENDER
protocol
query
answer
ROM
MANAGER
protocol
query
answer
OPQ 1
OPQ N
IPQ
LINK LAYER
CONTROLLER
LINK 1
ROM
PROTOCOL
STATE
FDBL
LINK LAYER
CONTROLLER
LINK N
L2q_attn



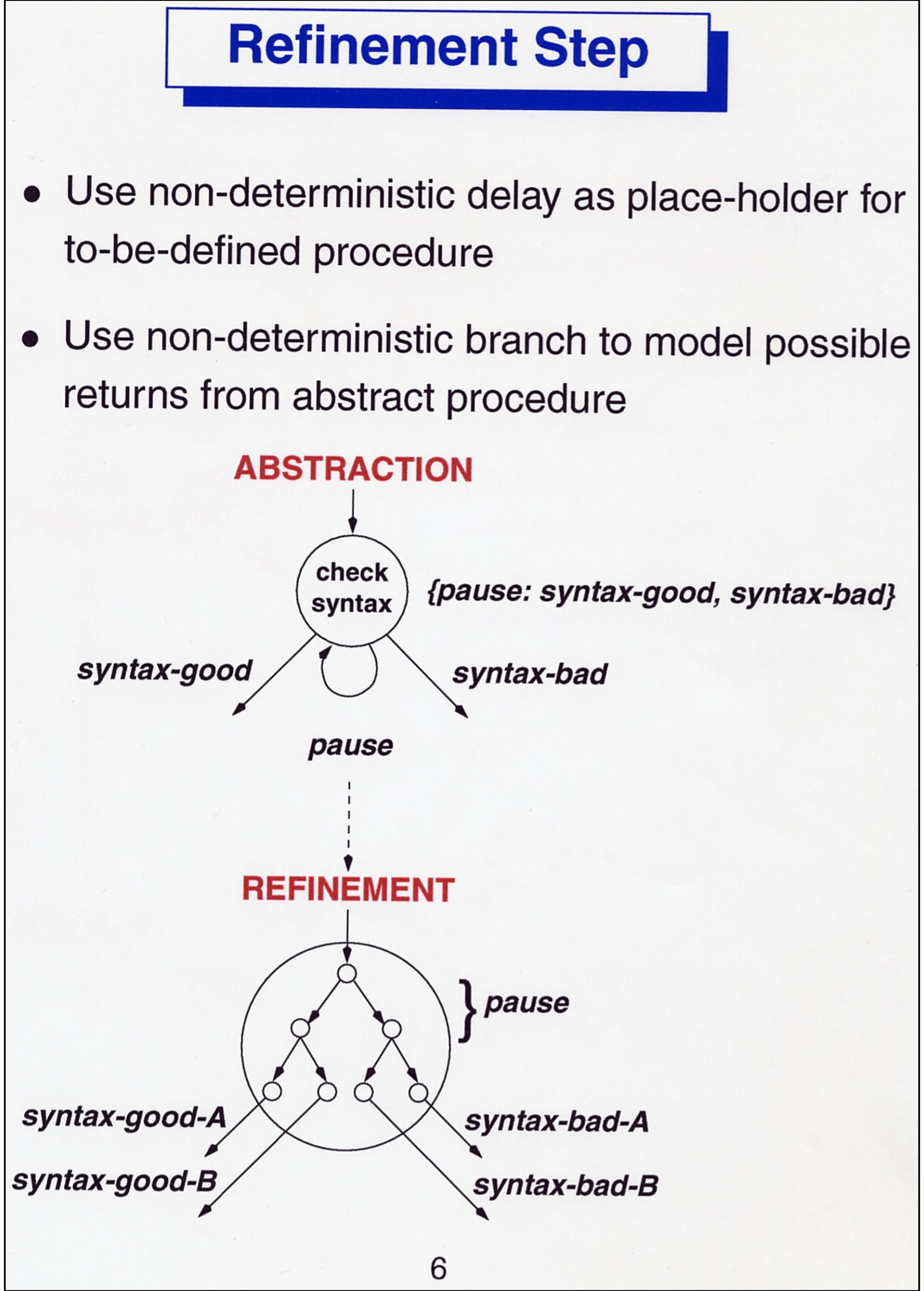
# Abstraction

subroutine
begin
return (answer)
{yes, no}
delay
abstraction
delay
begin
true
true
true
yes
no

Abstraction is more general than subroutine

- e.g., abstraction may never terminate

ALL FOLLOWING APPLIES TO autom. lang. containment, or CTL w̄ ∃. Explain why it doesnt apply to CTL with ∃.



## Refinement Step

- Use non-deterministic delay as place-holder for to-be-defined procedure
- Use non-deterministic branch to model possible returns from abstract procedure

**Conservative abstraction of refinement: verify property only once, at highest level it's defined**

[Stub types](#)

Data

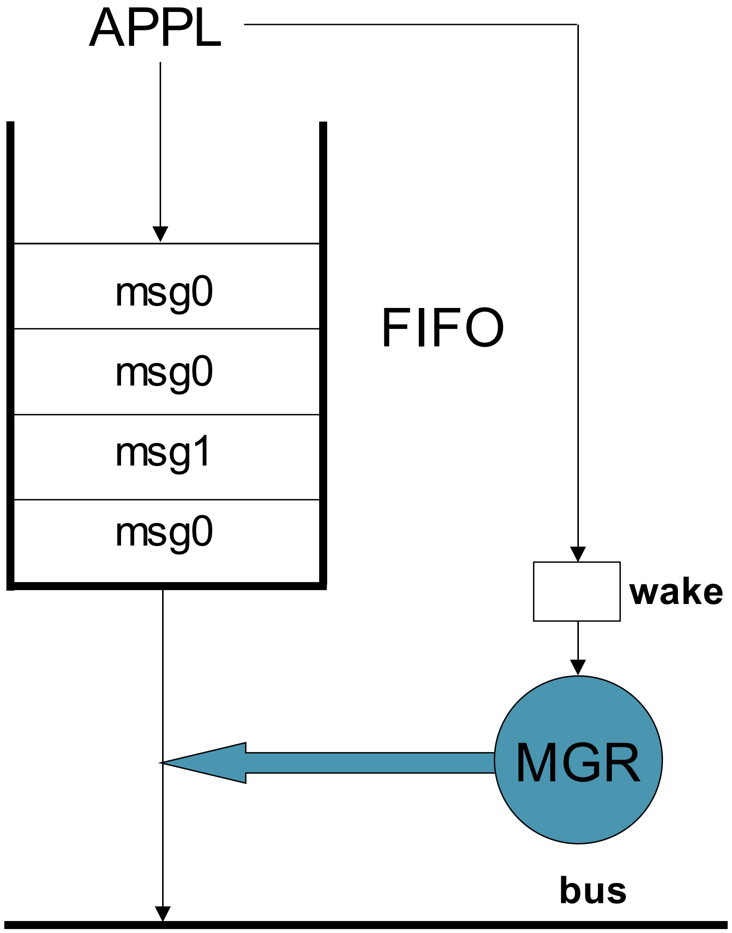
Datapath

Control

datapath = datastructure

control = FSM

# Example: stubbing a FIFO – Lv1



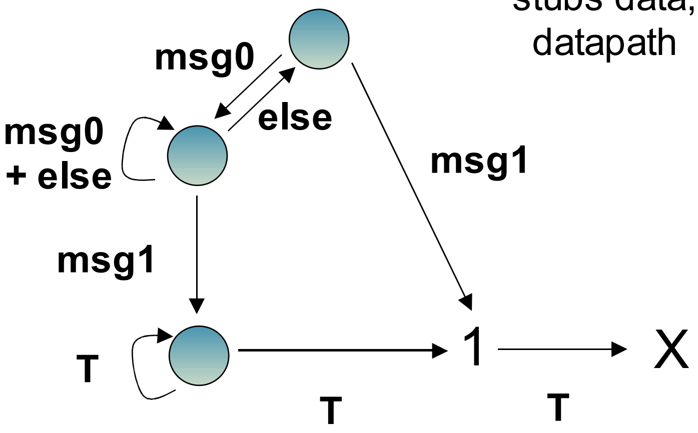
Lv1 data abstraction: track msg1,
All others -> msg0

-----------------------------------------------

Lv1 Assertion: After (*APPL.put_msg1*)
Eventually(*msg1_on_bus*)
[verifies MGR]

## FIFO STUB

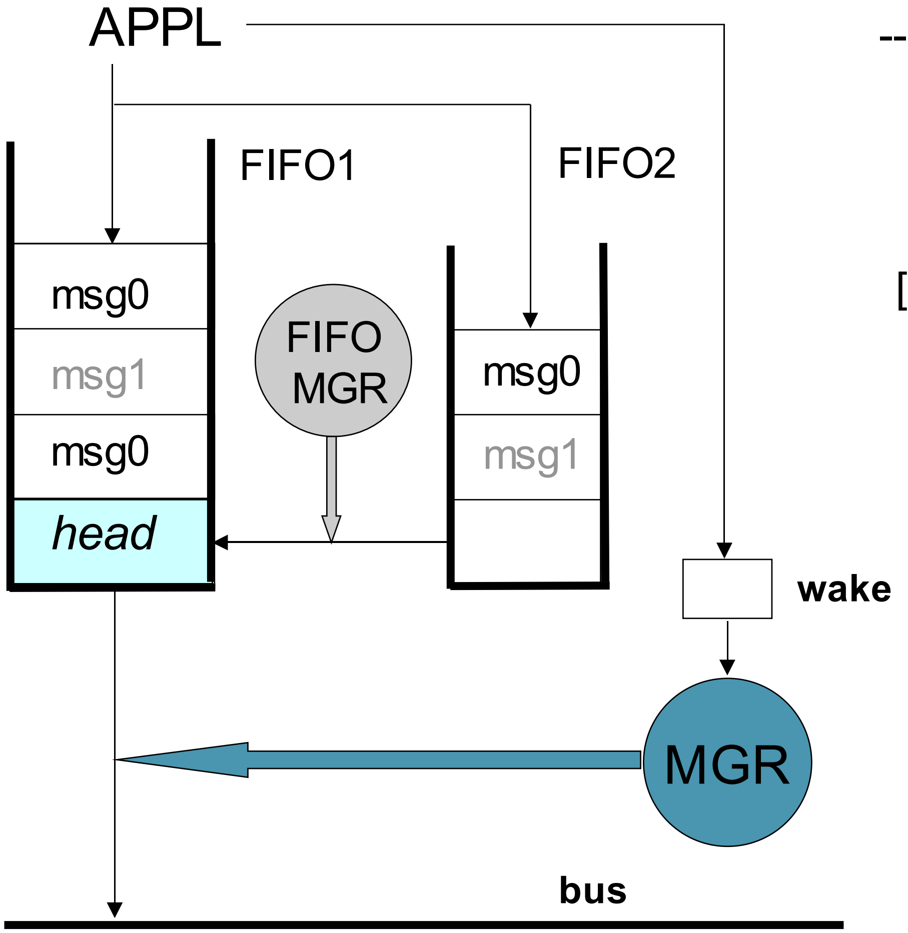
Lv1 Constraint: After (*FIFO.tail=msg1*)
Assume Eventually (*FIFO.head=msg1*)



stubs data,
datapath

thanks Chris Komar

# Example: stubbing a FIFO – Lv2



Lv2 refines FIFO stub into
2 sub-stubs
single msg1 can enter either

Lv2 Assertion: After (*FIFO.tail=msg1*)
Eventually (*FIFO.head=msg1*)
(= Lv1 constraint)
[Checks that FIFO MGR prevents starvation]

## FIFO STUB

Lv2 Constraints:
After (*FIFO1.tail=msg1*)Assume Eventually
(*FIFO1.head-1=msg1*)
After (*FIFO2.tail=msg1*)Assume Eventually
(*FIFO2.head=msg1*)

thanks Vic Du

# Further Refinements

Lv3: add FIFO mechanism (head/tail pointers)

- verify succession for real stages + abstract stage abstracting any number of words

  (verifies Lv2 constraints)

Lv4: expand abstract stage to full length of FIFO

- succession property follows inductively

Lv5: expand stages to full word width

- succession property follows inductively

# Consequences

- Design and verification done together

  → earlier hence cheaper debug

  -- D sees bugs as they're encoded (not months later)

  -- debug when design is simpler, hence easier to fix (fewer adjacent consequences)

- PV promoted to S/VE
- D designs global flow control before low-level data structures (iteratively)

  →Designer focuses on function before structure

  -- structure serves function (today it's reverse)

  eg, requirements for memory coherence will precede and define requirements for a cache protocol (rather than reverse)

- Coverage/Capacity scales linearly with design size

QUESTIONS?