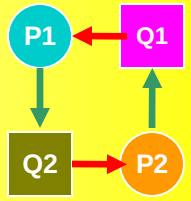


Hardware/Software Deadlock Avoidance for Multiprocessor Multiresource System-on-a-Chip

Dissertation Defense

By

Jaehwan Lee

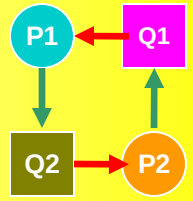
Advisor: Vincent J. Mooney III

School of Electrical and Computer Engineering

Georgia Institute of Technology
Atlanta, GA USA

Georgia Tech

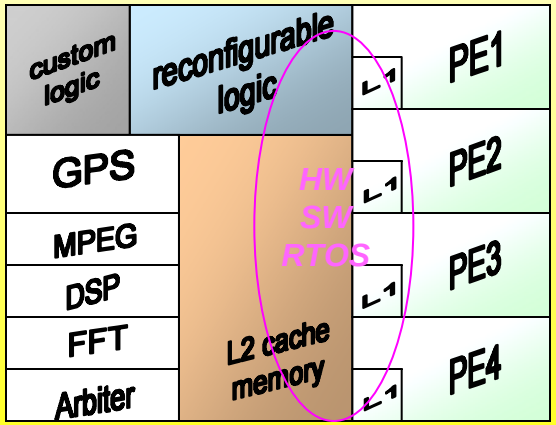
Outline



- Motivation and Objective
- Background and Prior Work
- Proofs about Deadlock Detection Unit
- Deadlock Avoidance Unit
- Parallel Banker's Algorithm Unit
- Integration into δ hardware/software RTOS framework
- Conclusion

Motivation
- Technology Trends

- Future SoC's
  - Multiple heterogeneous processors (tens of processes)
  - Multiple on-chip hardware resources
    - DSP, FFT, MPEG, GPS, Shared Memory, etc
  - Examples
    - Xilinx Virtex-II Pro FPGA includes multiple PowerPC
    - Broadcom BCM1400 includes multiple MIPS64
- Processes in such an SoC
  - Dynamically request and use resources
  - May end up in deadlock
- Current embedded system or single processor system
  - Typically ignored today



SoC: System on Chip

Georgia Tech

Motivation
- Does deadlock really matter?

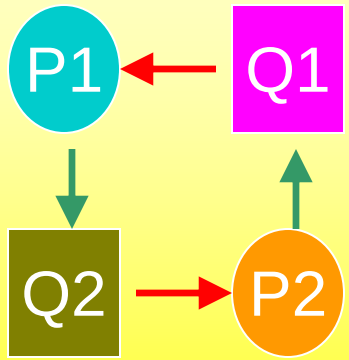
- Examples of future real-time systems
  - Human-like robot with multiple processes
    - In case of deadlock
      - Damage
      - People get injured
      - Law-suits
  - Mars Rover
    - In case of deadlock
      - Money loss
      - Time loss
  - Industrial control
  - Cars
- Even if low probability of deadlock, prefer no deadlock whatsoever

Objective
- Problem, Solution and Goal

- Problem
  - How to deal with deadlock?
- Goal
  - Allow software to make requests in any order
  - Grant as many resources as possible
  - Avoid deadlock correctly and quickly
- Solution
  - A hardware/software mechanism of deadlock avoidance, easily applicable to Real-Time Multiprocessor System-on-a-Chip (SoC) design

Background: Definitions

- Definition of Deadlock
  - A system has a deadlock iff the system has a set of processes, each of which is blocked, waiting for requirements that can never be satisfied



Background: Definitions

- Definition of Livelock
  - Livelock is a situation where a request for a resource is repeatedly denied and possibly never accepted because of the unavailability of the resource, resulting in a stalled process, while the resource is made available for other process(es) which make progress

Background: Definitions

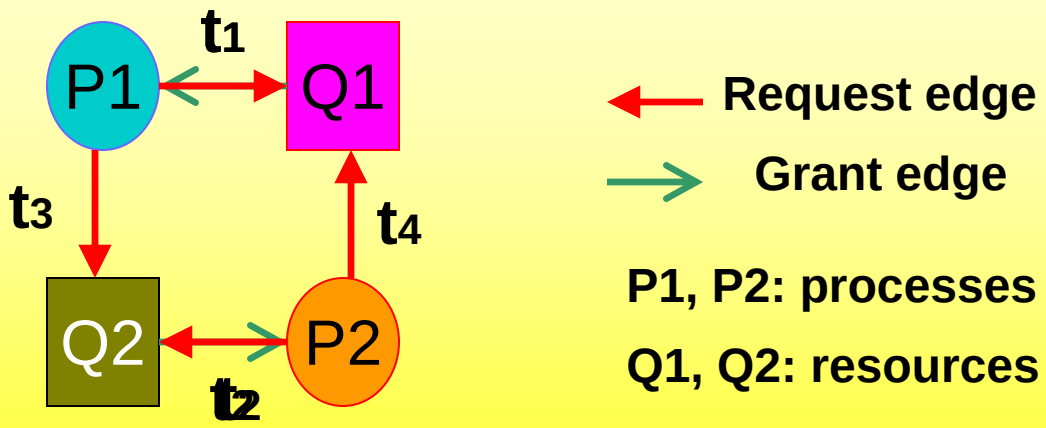
- Definition of Deadlock Avoidance
  - A way of dealing with deadlock where resource usage is dynamically controlled not to reach deadlock (i.e., on the fly, resource usage is controlled to ensure that there can never be deadlock)

Background: Definitions

- Definition of a Safe Sequence
  - A safe sequence is an enumeration p_1, p_2, ..., p_n of all processes in the system, such that for each i=1, 2, ..., n, the resources that p_i may request are a subset of the union of resources currently available plus resources currently held by p_1, p_2, ..., p_{i-1}

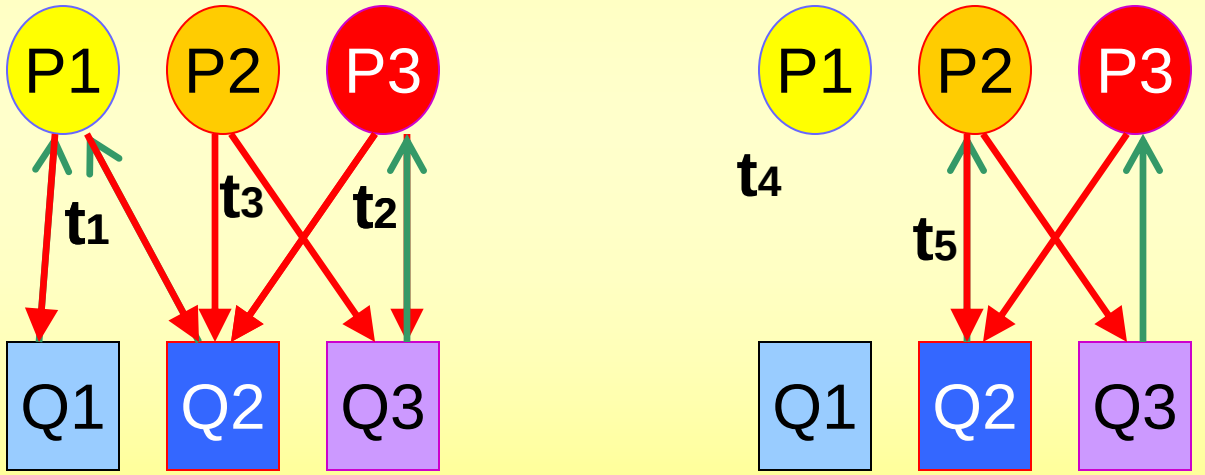
Background: Terms
Request deadlock (R-dl)

- A deadlock situation directly caused by a request
- Assumptions: No restriction on resource usage
  - (i) P1 requires either Q1, Q2, or both depending on software flow
  - (ii) P2 also requires either Q1, Q2, or both
  - (iii) We don't know in advance
- When P1 and P2 take flows that they require both Q1 and Q2



Background: Terms
Grant deadlock (G-dl)

- A deadlock situation directly caused by a grant
  - The same assumptions with the previous



Request edge

Grant edge

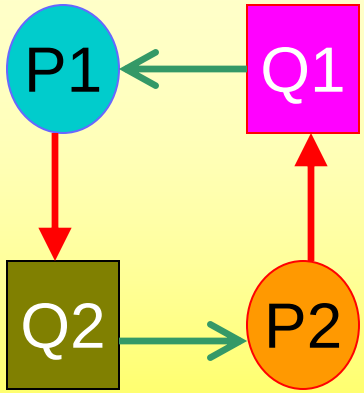
Assumption (iv):

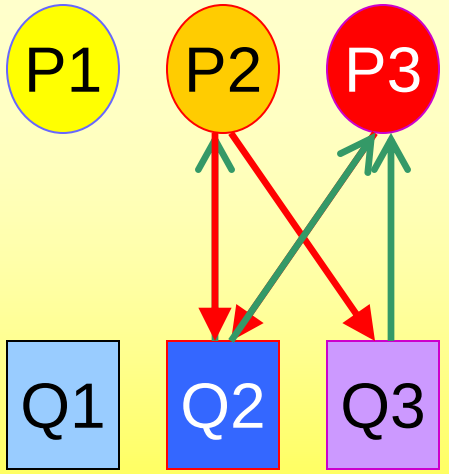
Priorities P1 > P2 > P3

Differentiation between R-dl and G-dl

- Reason
  - Some actions can only be taken for either R-dl or G-dl.

    (E.g., G-dl could have been avoided by granting Q2 to P3 instead of P2 in the previous G-dl example.)





Why does deadlock occur? - Four Deadlock Conditions

Properties of Resources

- Mutual Exclusion
  - No simultaneous sharing of a resource
- No Preemption
  - A resource can be released only by the process holding the resource.

Behavior of Processes

- Hold and Wait
  - A process may hold some resources while the process requests additional resources.
- Circular Wait
  - A process must wait for unavailable resources to become available.

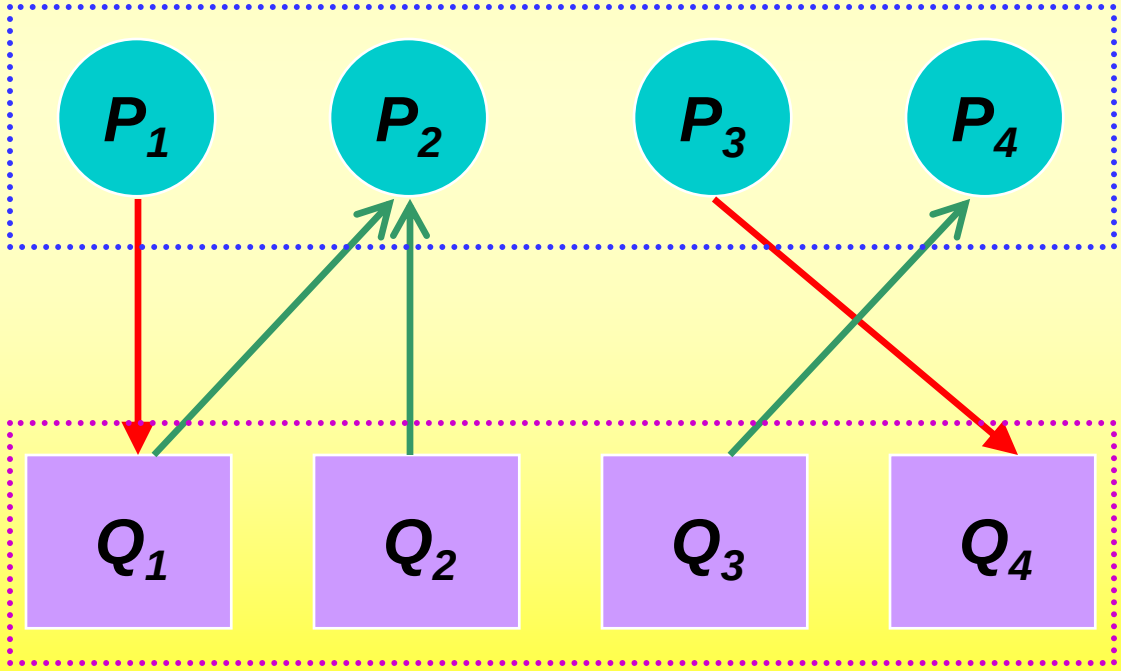
Background: Terms
Single vs. Multiple Instance Resources

- Single instance resources
  - A resource that can support one process at a time
    - E.g., a printer
- Multiple instance resources
  - A resource that can support a certain number of multiple processes simultaneously
    - E.g., a counting semaphore associated with allocable memory

Background: Terms

Resource Allocation Graph (RAG)

- A Set of Nodes V = P U Q
  - P_i: processes, Q_j: resources
- A Set of Edges E = $(P_i \rightarrow Q_j)$ U $(Q_j \rightarrow P_i)$



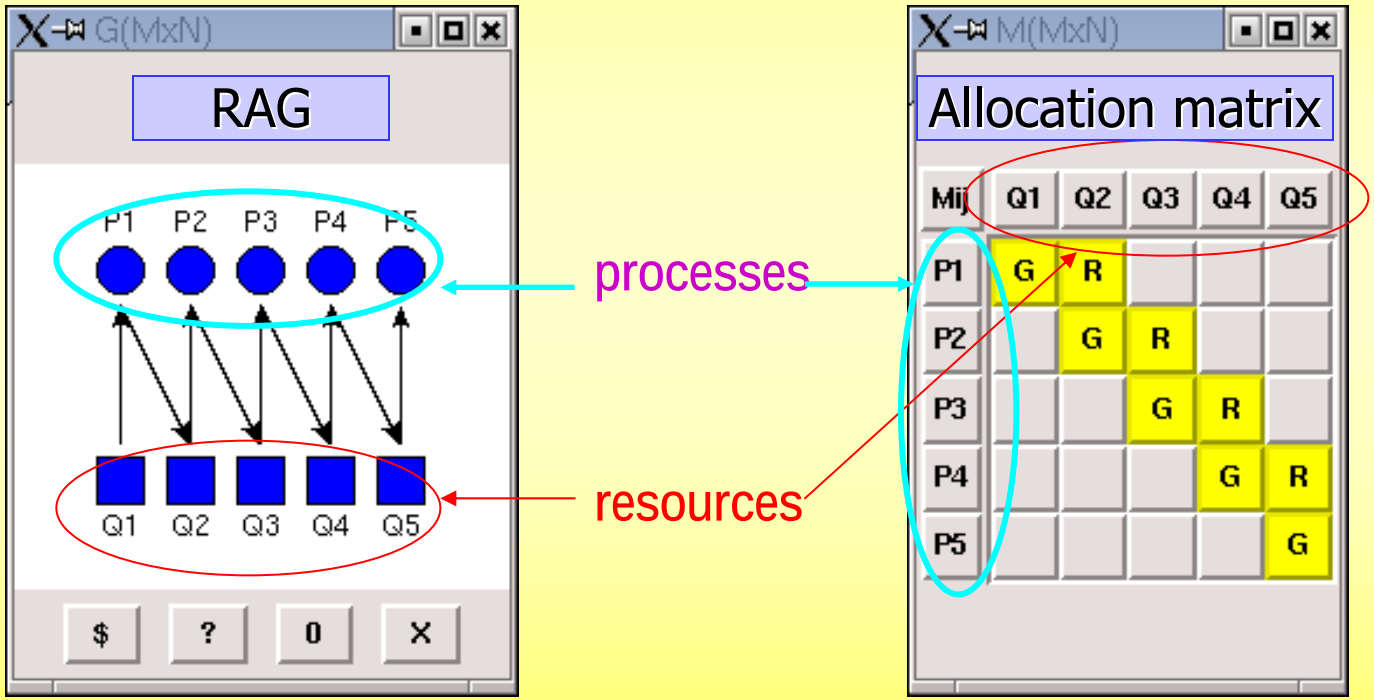
Request edge

Grant edge

"Bipartite Graph"

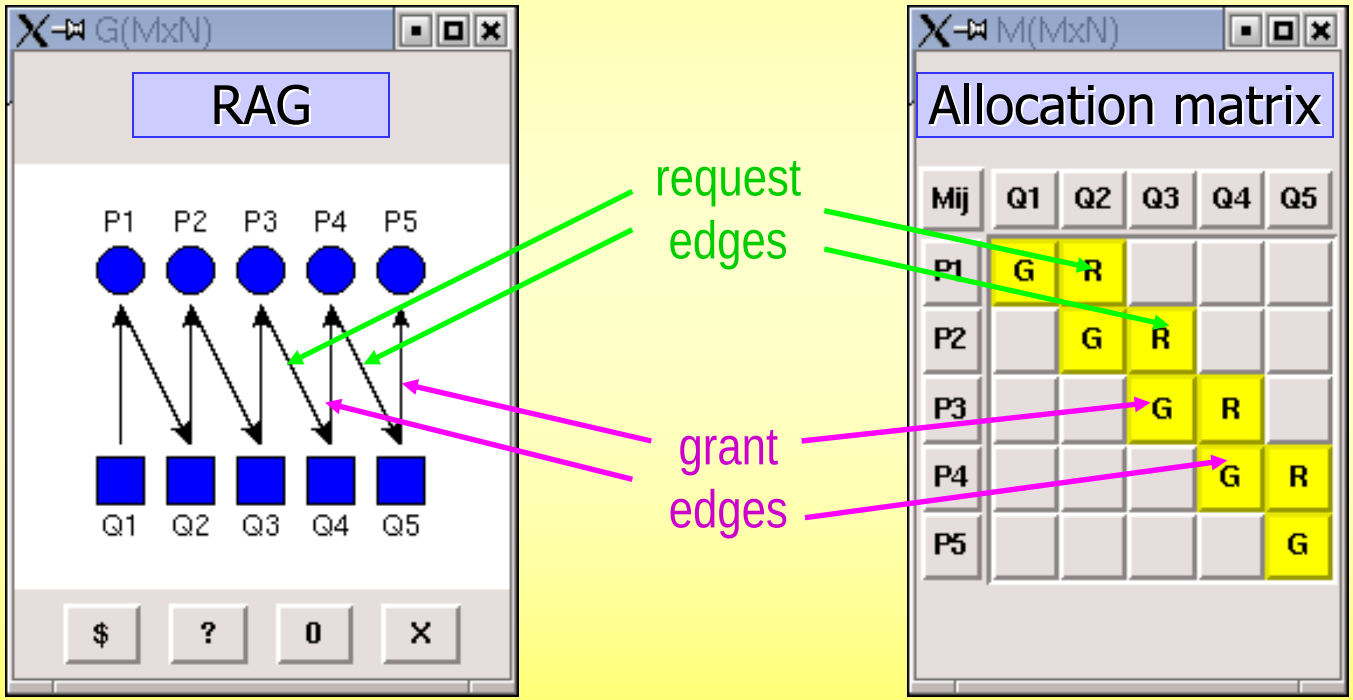
Terms – RAG and Corresponding Matrix Representation M

- Used in Deadlock Detection Unit (DDU)

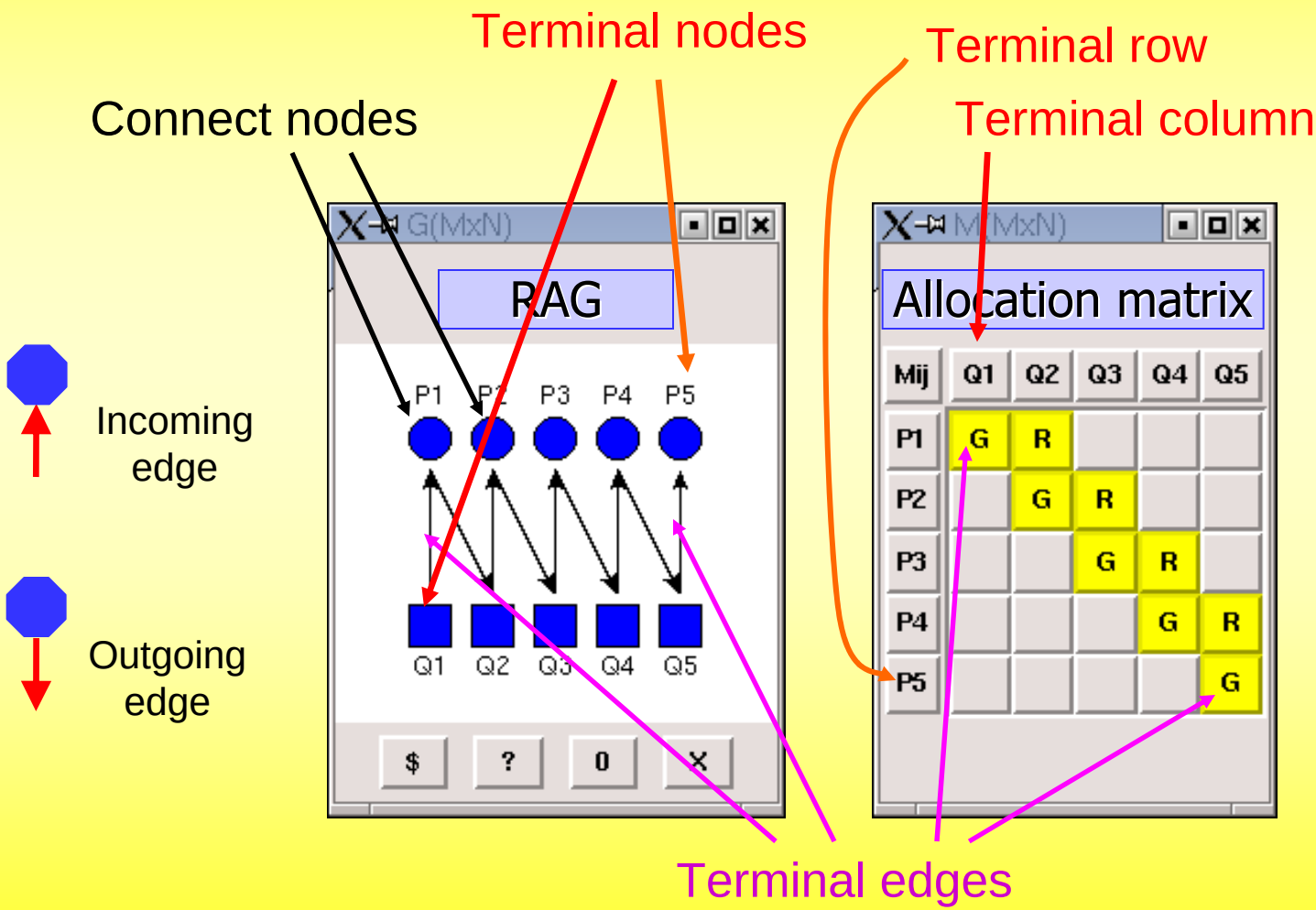


Terms – RAG and

Corresponding Matrix Representation M



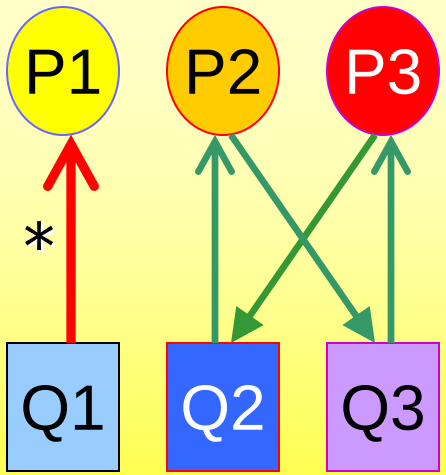
Terms – RAG and Corresponding Matrix Representation M



Georgia Tech

Prior Work by Shiu, Tan and Mooney
Deadlock Detection Hardware Unit (DDU)

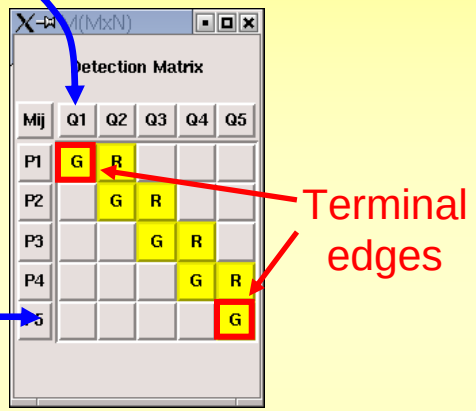
- Matrix based parallel operation approach
- Terminal edge reduction technique to reveal cycles (i.e., deadlock)
  - Terminal (i.e., removable) edge*: not related to deadlock
- Simple bit-wise Boolean operations
  - Implementation easier
  - Operation faster, O(min(m,n))
  - 2~3 orders of magnitude faster than software
- Novelty from previous algorithms
  - Does NOT trace exact cycles
  - Does NOT require linked lists

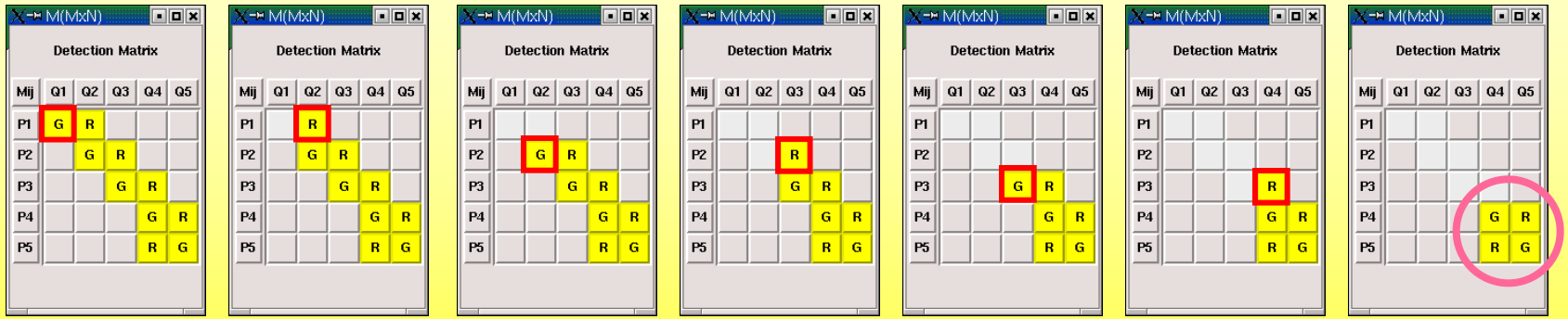


P. Shiu, Y. Tan and V. Mooney, "A Novel Parallel Deadlock Detection Algorithm and Architecture," *9th International Workshop on Hardware/Software Codesign* (CODES'01), pp. 30-36, April 2001.

Abstraction of DDU

- Check the reducibility of an allocation matrix
  - Remove all terminal edges currently revealed at each step in matrix M
    - If ∃(either R's or G) in column i (terminal column)
      ⇒ remove all entries in column i
    - If ∃(either R's or G's) in row j (terminal row)
      ⇒ Remove all entries in row j
  - Iterate until ¬(reducible) or empty
- Determine deadlock
  - If empty (no edges), no deadlock
  - If ¬(empty), deadlock





Iterations → Not empty → Deadlock

Georgia Tech

Prior Work in Deadlock Avoidance

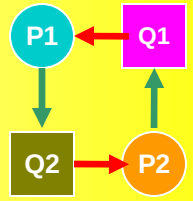
- Traditional Methods
  - Require some knowledge of future requests
  - Declare maximum claims (each process)
  - Give a grant only if remaining in a safe state
- Advantages
  - No deadlock
  - No preemption
- Disadvantages
  - Low resource utilization
    - Worse for single unit resources
  - Performance degradation
    - Due to dynamic avoidance decision
  - Practical issues – long software run-time

Prior Work in Deadlock Avoidance

- Dijkstra: Banker's Algorithm (1968)
  - For a single multiple-instance resource
  - Maximum claims strategy
- Habermann: $O(nm^2)$ (1969)
  - For multiple-instance multiple resources
  - Livelock problem
- Holt: O(mn) (1972)
  - Solved livelock problem using wait-time counters
  - For general resource systems
- Belik (1990)
  - Path matrix representation
    - Resource allocation →
      Changing an acyclic digraph while keeping it acyclic
  - For multiple single-instance resources
  - O(mn) in general, O(1) in the best case

- No prior work in hardware implementation of a deadlock avoidance approach

Outline



- Motivation and Objective
- Background and Prior Work
- Proofs about Deadlock Detection Unit
- Deadlock Avoidance Unit
- Parallel Banker's Algorithm Unit
- Integration into δ hardware/software RTOS framework
- Conclusion

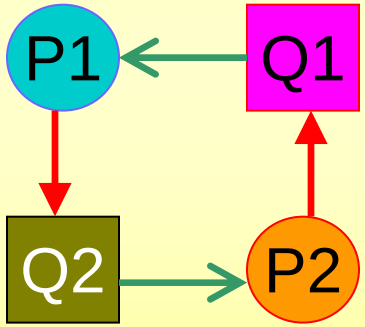
Proofs of the Correctness of and run-time complexity of DDU

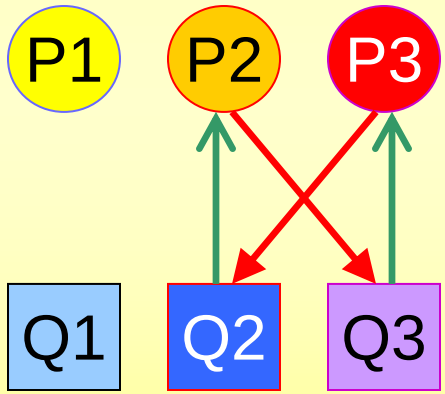
- Why do proofs matter?
  - Authenticity
    - No false alarm
    - To avoid any damages or liability
  - To ensure timeliness in a real-time SoC
    - Within a certain amount of time for a robot not to fall over in a deadlock

* Additional details can be found in a journal submission and a technical report [3, 7]

Proof of the Correctness of DDU

- Deadlock vs. Cycle Relation
  - Deadlock $\Rightarrow \exists$ Cycle(s) in an RAG
  - Cycle(s) $\Rightarrow \exists$ Deadlock

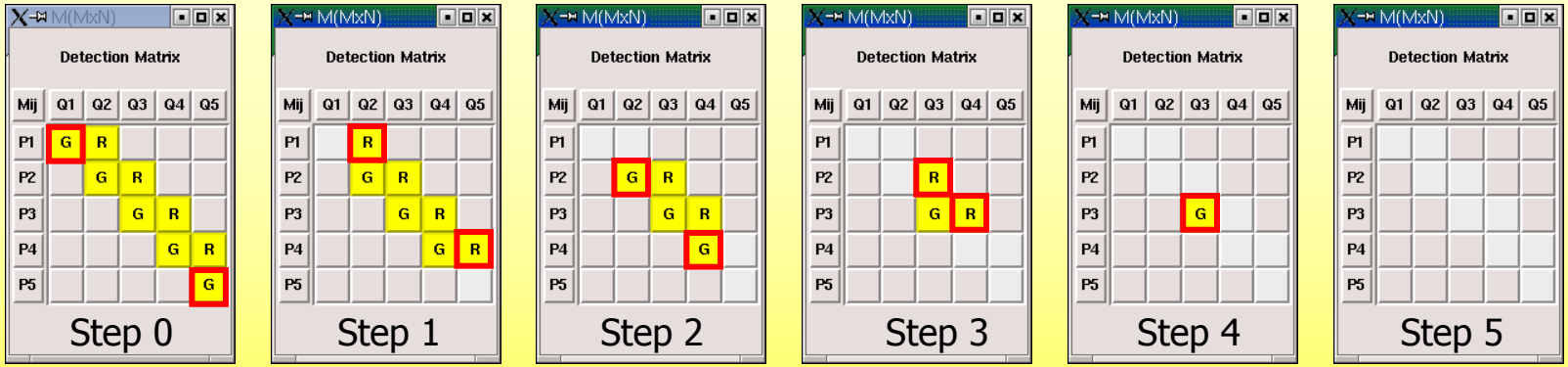




- DDU detects deadlock iff there exists a cycle in an RAG

Proof of the Correctness of DDU - Intuition behind proofs

- Remove edges in columns and rows with only G's or only R's
  - Operations performed in parallel throughout the matrix at each iteration
- A cycle: a circular pairs of R-G edges in rows and in columns

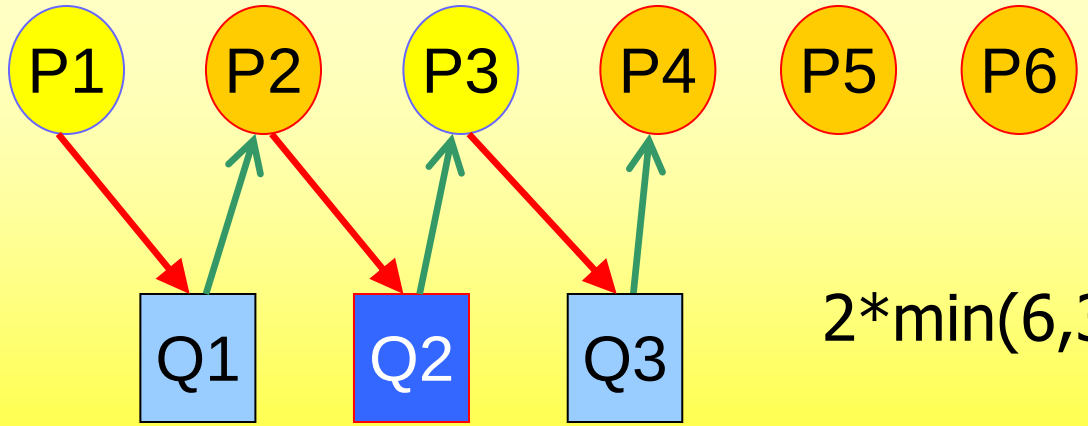


Iterations → Completely reducible → No deadlock

Georgia Tech

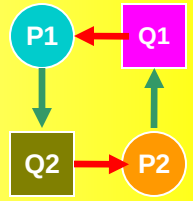
Proof of the Run-time Complexity of DDU

- An upper bound on the number of edges in a path (not a cycle or RAG) = 2*min(m,n)
  - Due to the bipartite property
- DDU, implemented in hardware, completes its computation in at most 2*min(m,n) - 3 steps
  - O(min(m,n))



* Additional details in a journal submission and a technical report [3, 7]

Outline



- Motivation and Objective
- Background and Prior Work
- Proofs about Deadlock Detection Unit
- Deadlock Avoidance Unit
- Parallel Banker's Algorithm Unit
- Integration into δ hardware/software RTOS framework
- Conclusion

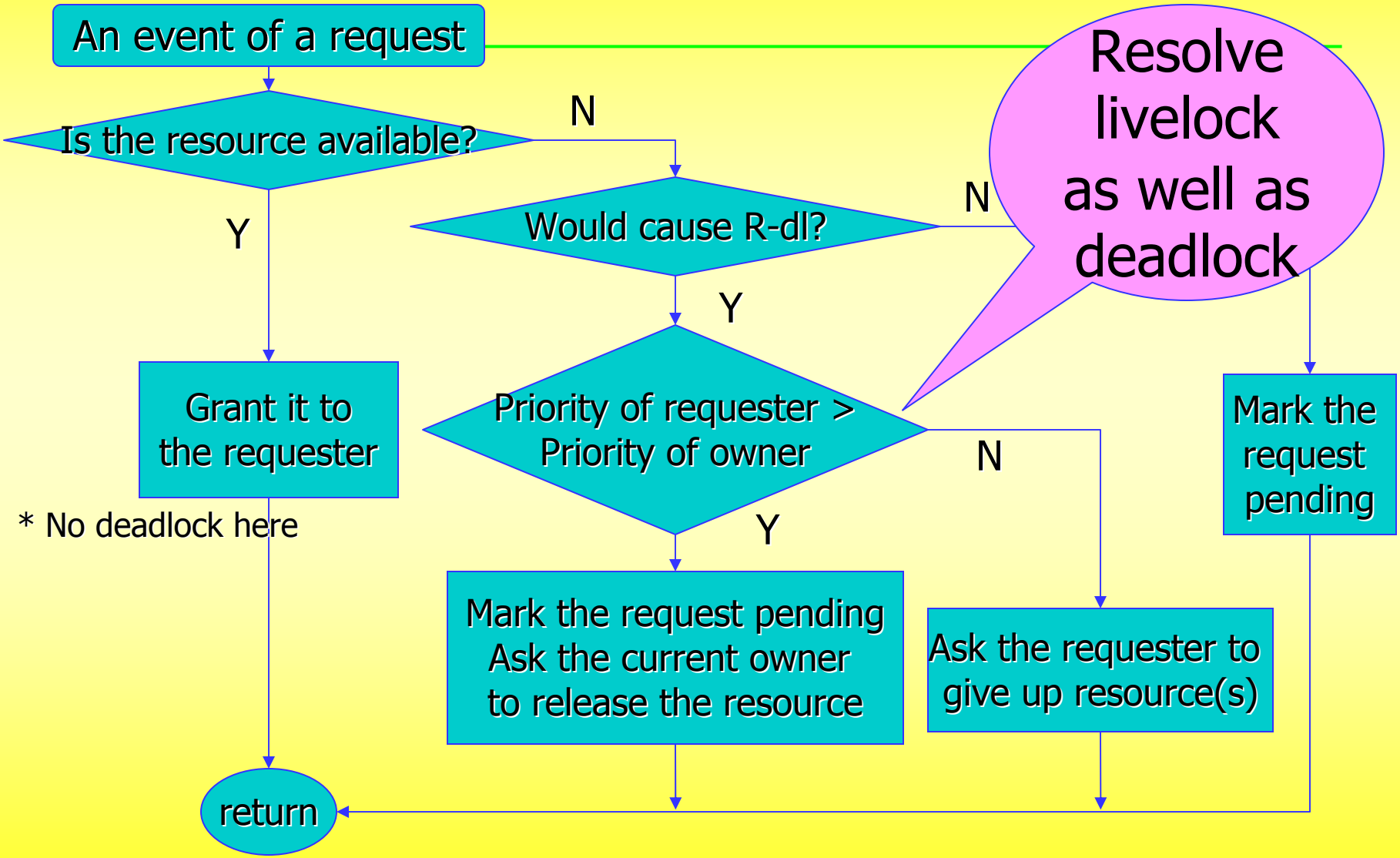
Deadlock Avoidance Unit (DAU)

- No declaration of maximum claims
- No restriction on resource usage
- Advantages
  - Higher resource utilization
  - Fast avoidance due to hardware implementation
- Disadvantages
  - Somewhat unfairness on a special occasion
    - When avoiding G-dl

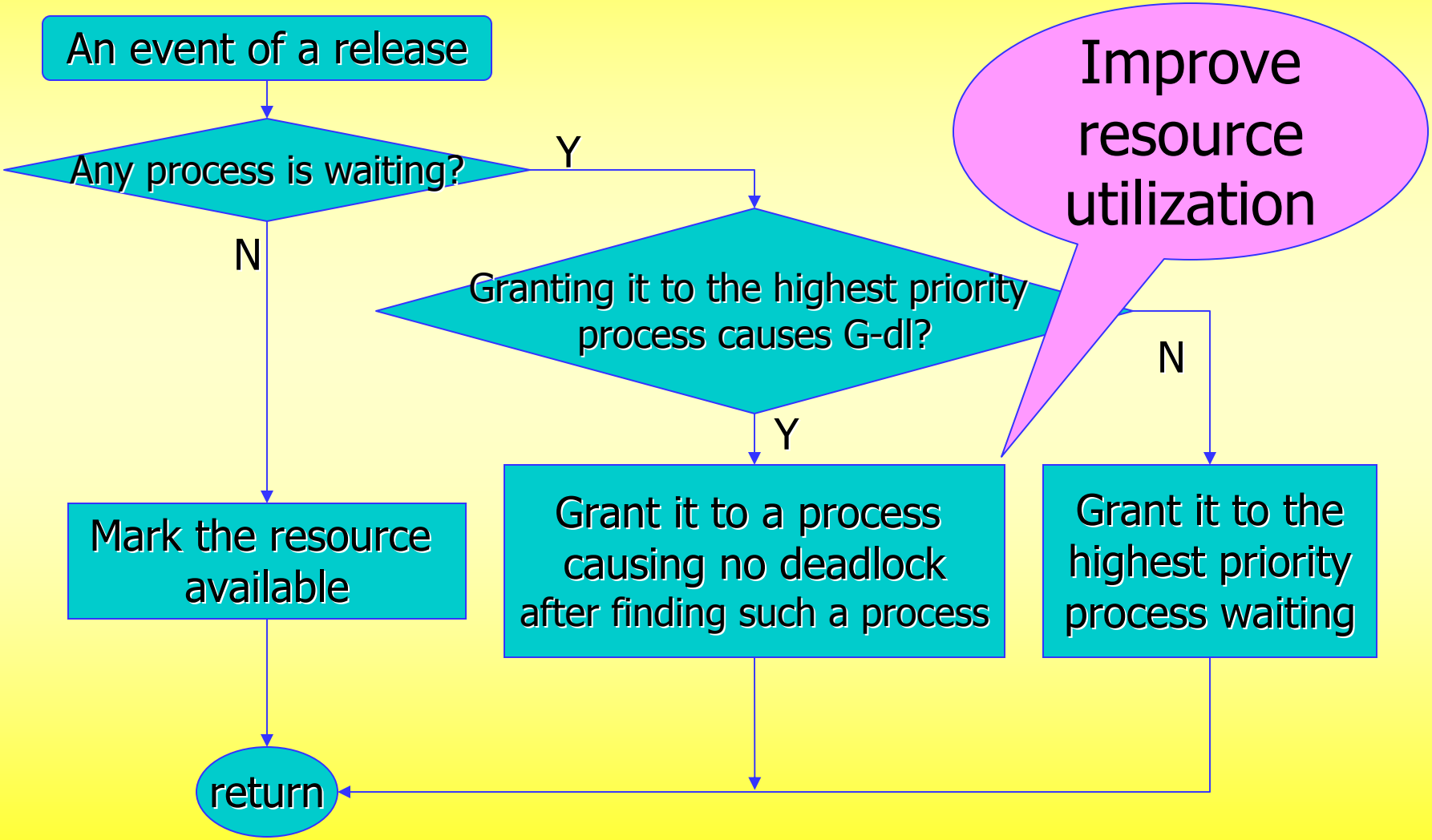
      (a lower priority process could proceed before a higher priority process, which would end up in deadlock)

      -- Similar concept to priority inheritance
    - But, resulting in higher resource utilization
  - Possibility of resource preemption
    - When avoiding R-dl
- Assumption: all resources single-instance

Deadlock Avoidance Algorithm (DAA)



Deadlock Avoidance Algorithm (DAA)

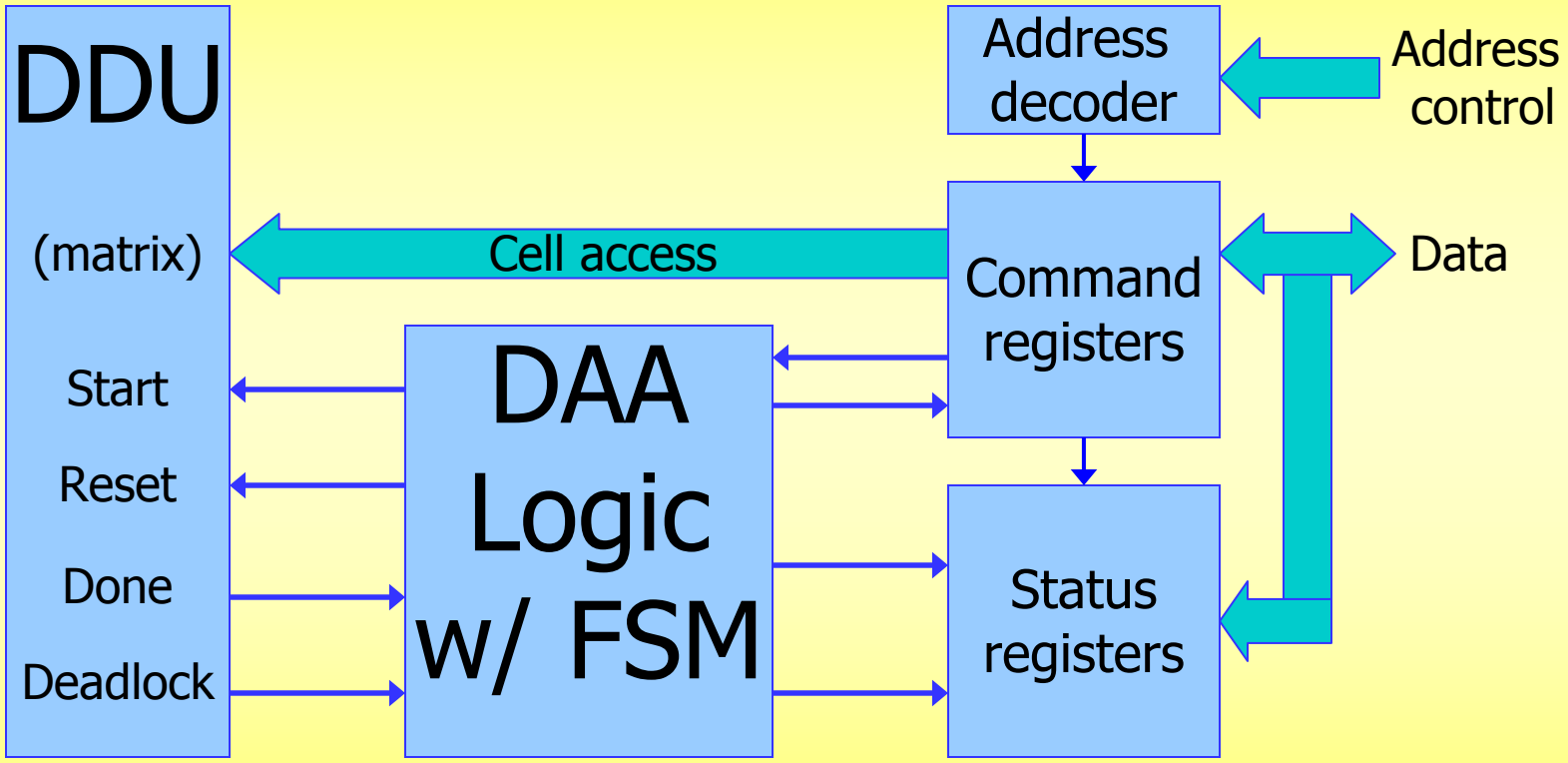


Proof of the Correctness of DAA

- Proof with lemmas and theorems
  - Theorem 1 (R-dl case)
    - Denying the request in the case of R-dl results in livelock unless a process involved in the deadlock releases a resource involved in the deadlock
  - Theorem 2 (G-dl case)
    - For a given system, not currently deadlocked, where a grant of a resource occurs, there must exist at least one process to which the resource can be granted without deadlock

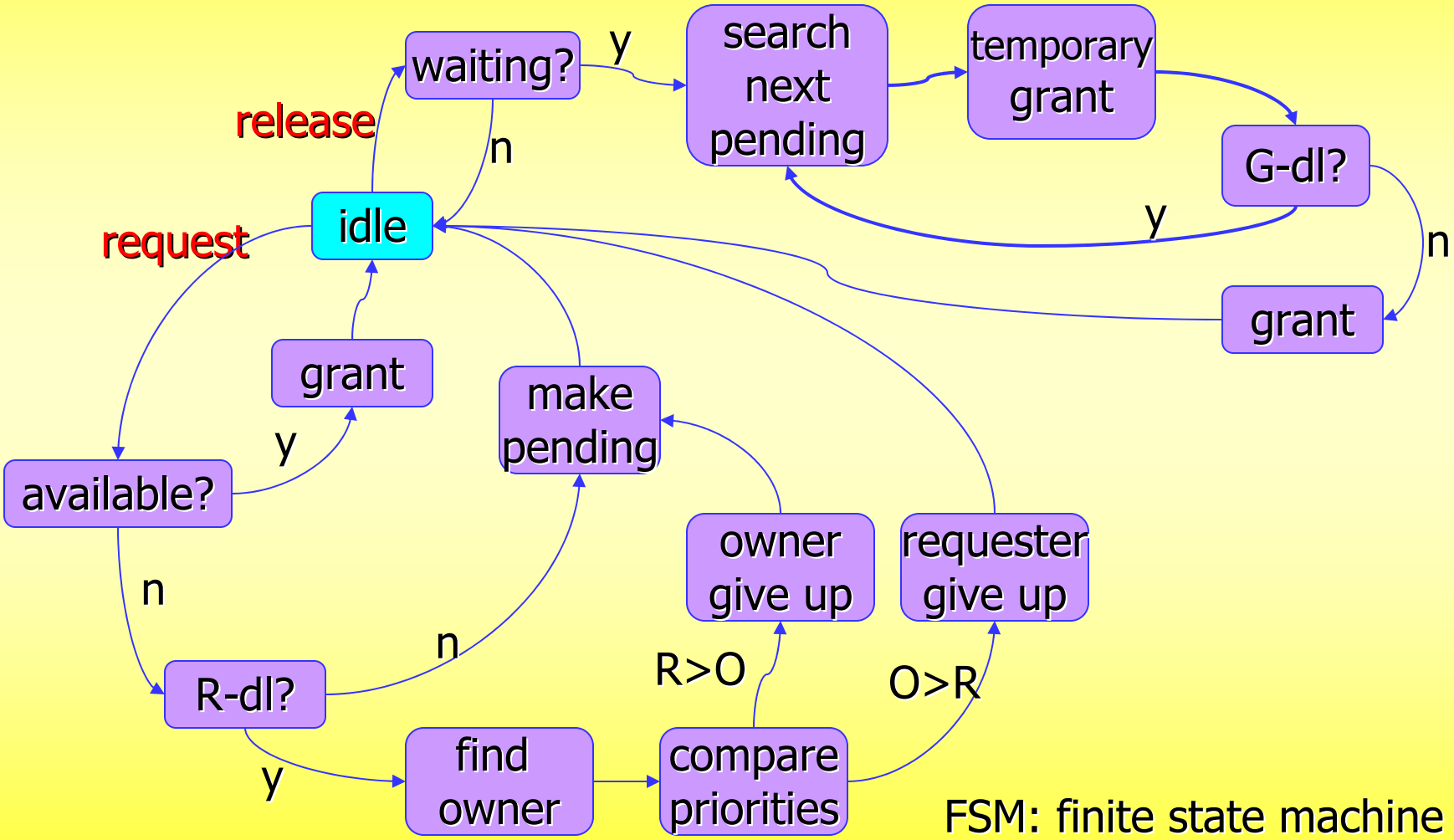
* Additional details in a technical report

Architecture of the DAU



DDU: Deadlock Detection Unit
FSM: finite state machine

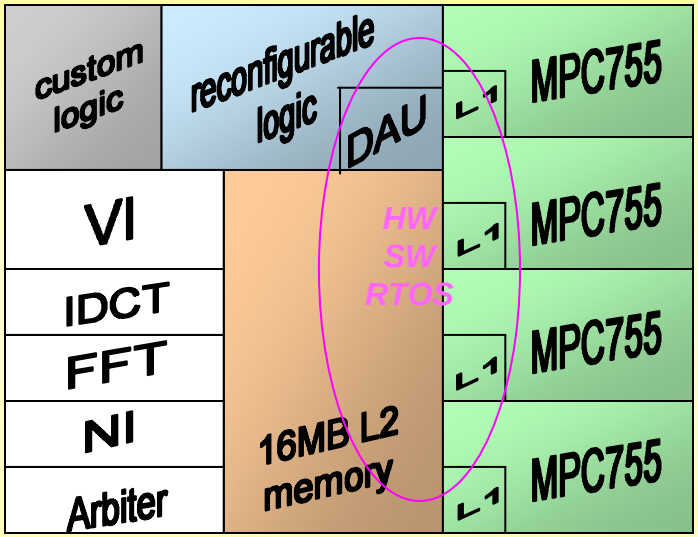
State Transition Diagram - DAA Logic FSM



FSM: finite state machine

DAU Experimentation
- Target System-on-a-Chip

- Four MPC755's
  - Each CPU has 32KB I-Cache and 32KB D-Cache
  - 100MHz external clock,
  - 16MB shared memory



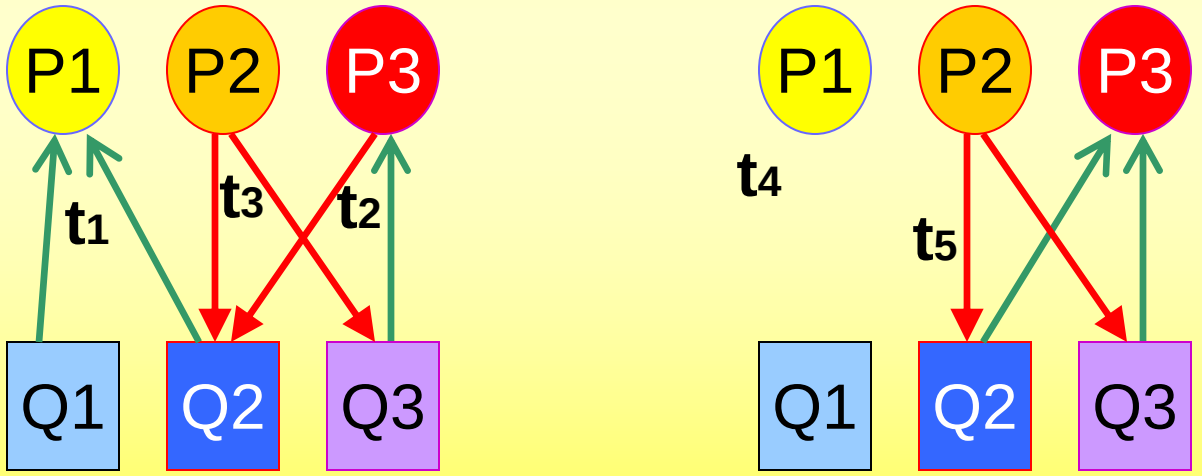
- Four resources
  - Q1: Video Interface (VI)
  - Q2: Inverse Discrete Cosine Transform (IDCT)
  - Q3: Fast Fourier Transform (FFT)
  - Q4: Network Interface (NI)

DAU Experimentation
- RTOS, Application, Environment

- Atalanta RTOS 0.3
  - By Sun, Blough and Mooney at Georgia Tech
- Each process requires two resources except P4
  - P1: processing a video stream (needs VI + IDCT)
  - P2: separating/enhancing frames (needs IDCT + FFT)
  - P3: extracting special images (needs FFT + VI)
  - P4: transferring images (needs NI)
  - One active process for each processing element (PE)
- Seamless CVE from Mentor Graphics
  - Instruction accurate simulation
  - VCS (Synopsys) and XRAY (Mentor)

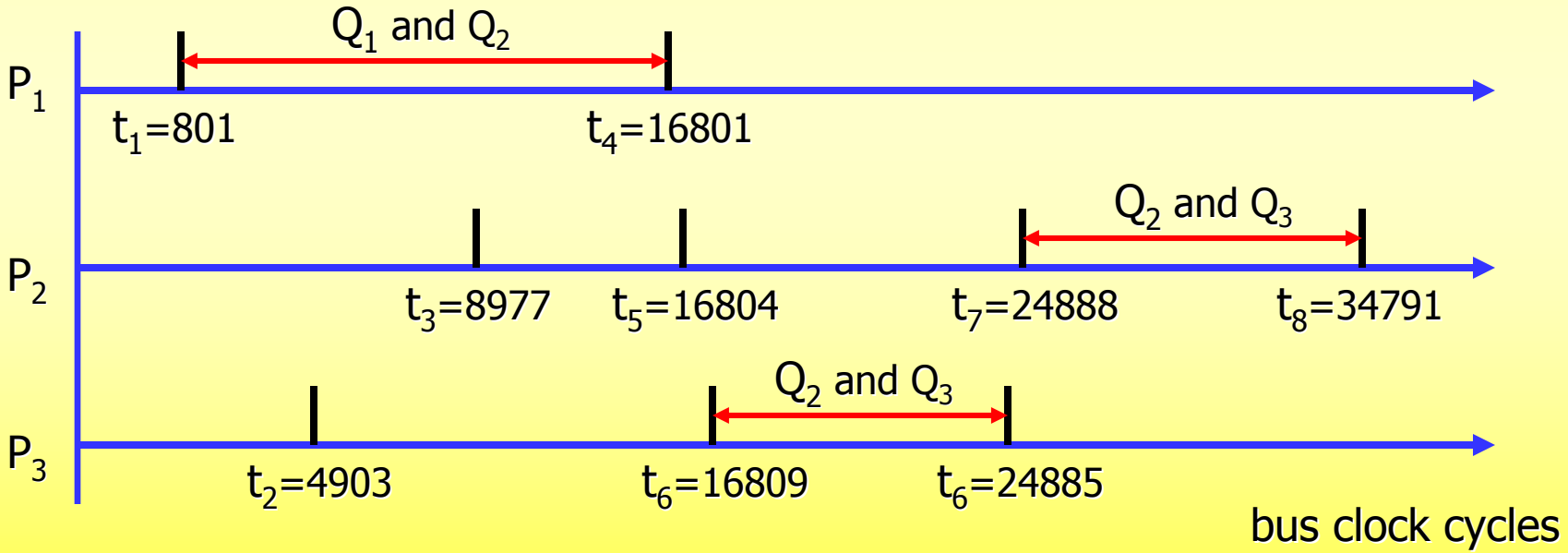
Experimental Results of DAU

- G-dl avoidance simulation: Two kinds
  - DAU: Synopsys VCS runs compiled Verilog code
  - DAA in software: MPC755 runs compiled C code in Seamless CVE
  - Example application invokes deadlock avoidance 12 times



Experimental Results of DAU (cont'd)

- G-dl avoidance simulation result
- Time line with resource usage



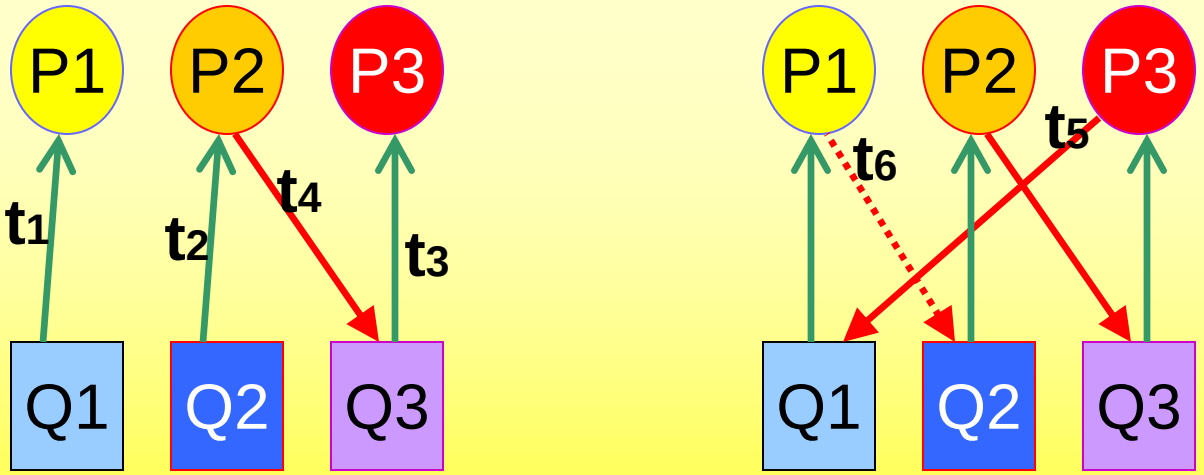
Experimental Results of DAU (cont'd)

- G-dl avoidance simulation result
- Performance improvement
  - 99% algorithm execution time reduction
  - 37% reduction in an application execution time

Method of Deadlock Avoidance	Algorithm Exe. Time (cycles)	Normalized Exe. Time	Application Exe. Time (cycles)
DAU Hardware	7 (average)	1X	34791
DAA in Software	2188 (average)	312X	47704

Experimental Results of DAU (cont'd)

- R-dl avoidance simulation: Two kinds
  - DAU: Synopsys VCS runs compiled Verilog code
  - DAA in software: MPC755 runs compiled C code in Seamless CVE
  - Example application invokes deadlock avoidance 14 times



Experimental Results of DAU (cont'd)

- R-dl avoidance simulation result
- Performance improvement
  - 99% algorithm execution time reduction
  - 44% reduction in an application execution time

Method of Deadlock Avoidance	Algorithm Exe. Time (cycles)	Normalized Exe. Time	Application Exe. Time (cycles)
DAU Hardware	7.13 (average)	1X	38508
DAA in Software	2102 (average)	294X	55627

Synthesis Results of DAU

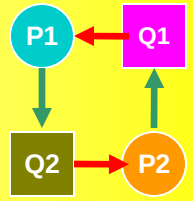
- Synopsys Design Compiler
- TSMC .25µm technology library from Qualcore Logic
- 0.05% of the total SoC area with four PEs and memory

Module Name	Total Area in terms of two-input NAND gates	Lines of Verilog HDL Code
DAU 5x5	1597	523
7x7	2429	552
10x10	4309	612
15x15	8868	753
20x20	15247	943
MPSoC	40 Million	-

Clock period used: 4 ns

TSMC: Taiwan Semiconductor Manufacturing Company
PE: Processing Element

Outline



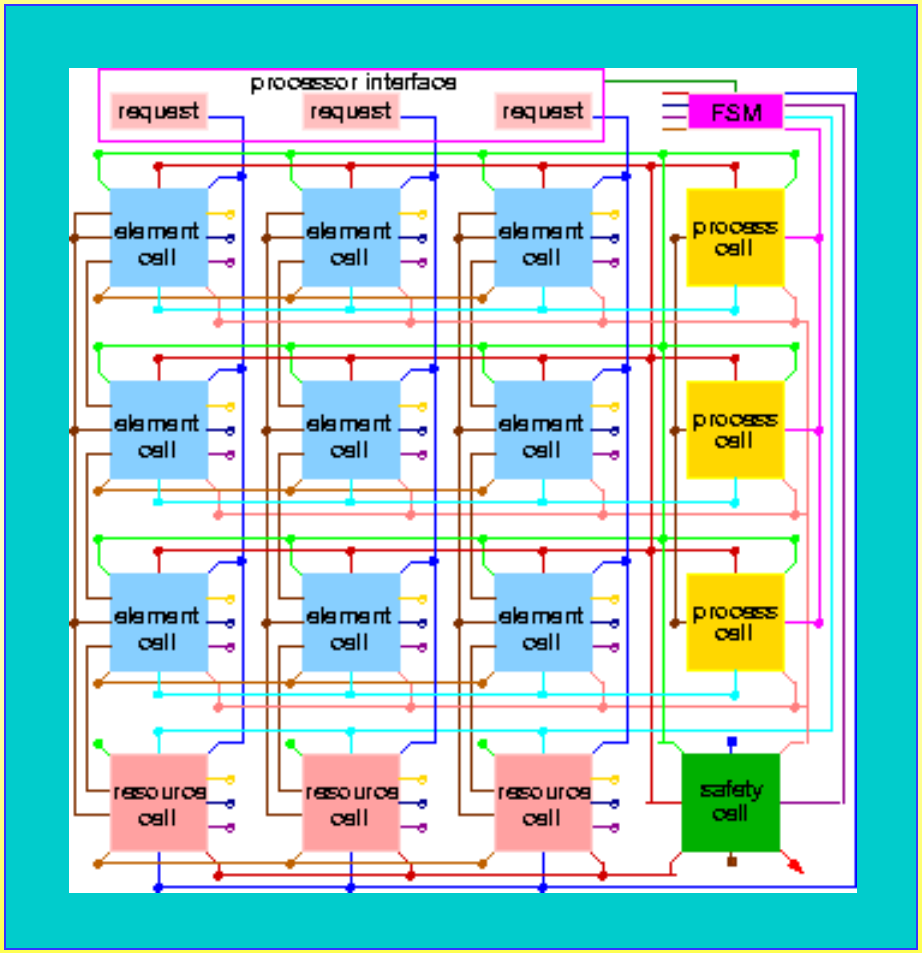
- Motivation and Objective
- Background and Prior Work
- Proofs about Deadlock Detection Unit
- Deadlock Avoidance Unit
- Parallel Banker's Algorithm Unit
- Integration into δ hardware/software RTOS framework
- Conclusion

Georgia Tech

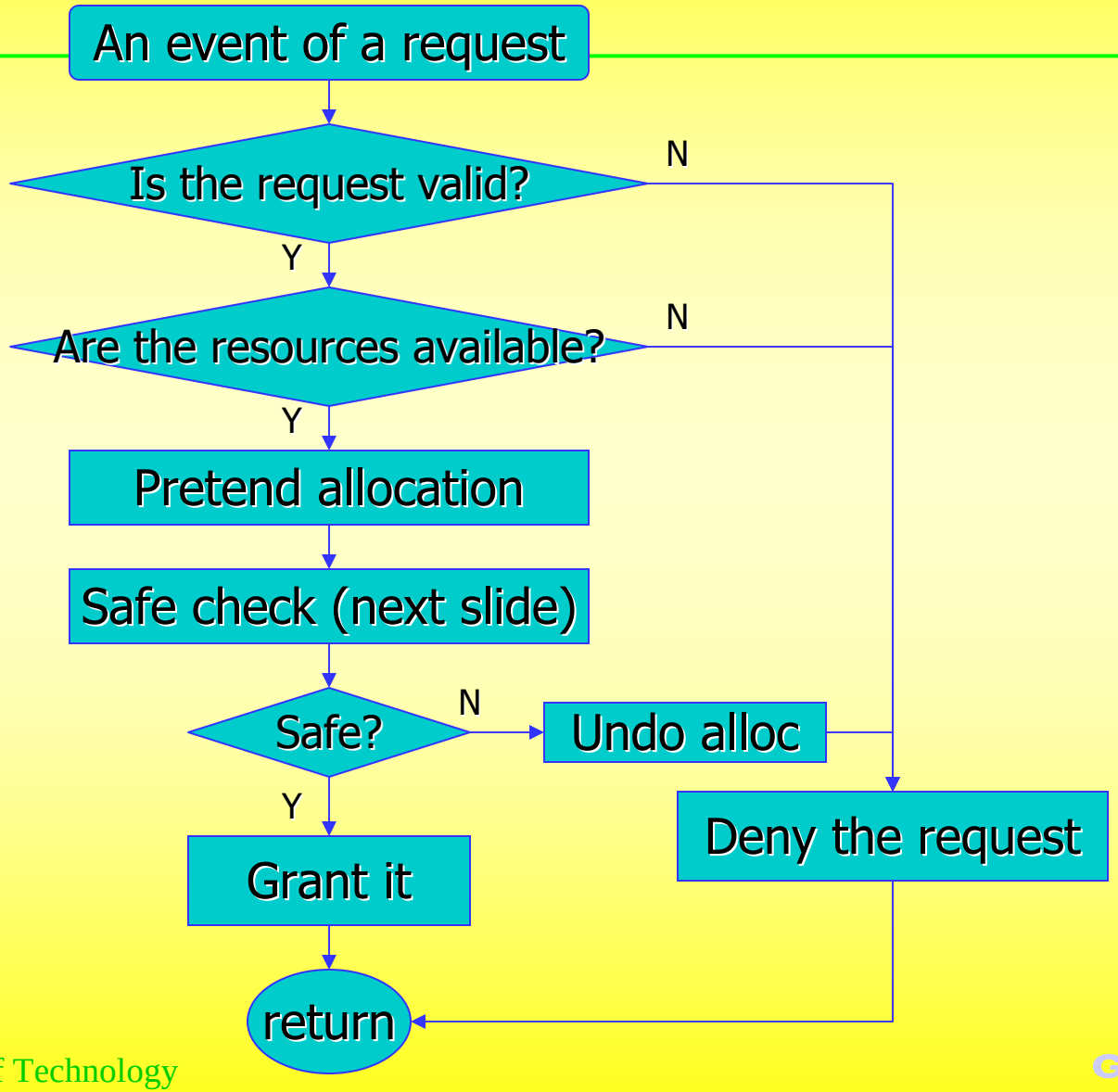
Parallel Banker's Algorithm (PBA)

- Parallelized Version of the Banker's Algorithm
  - For multiple-instance resources
- Advantages
  - Guarantee deadlock avoidance
  - Support multiple instance resources
  - Provide O(n) run-time complexity
    - Reduced from the original $O(mn^2)$
    - O(1) in the best case
- Disadvantages
  - Require hardware
  - Require maximum claim declaration
  - May under-utilize resources

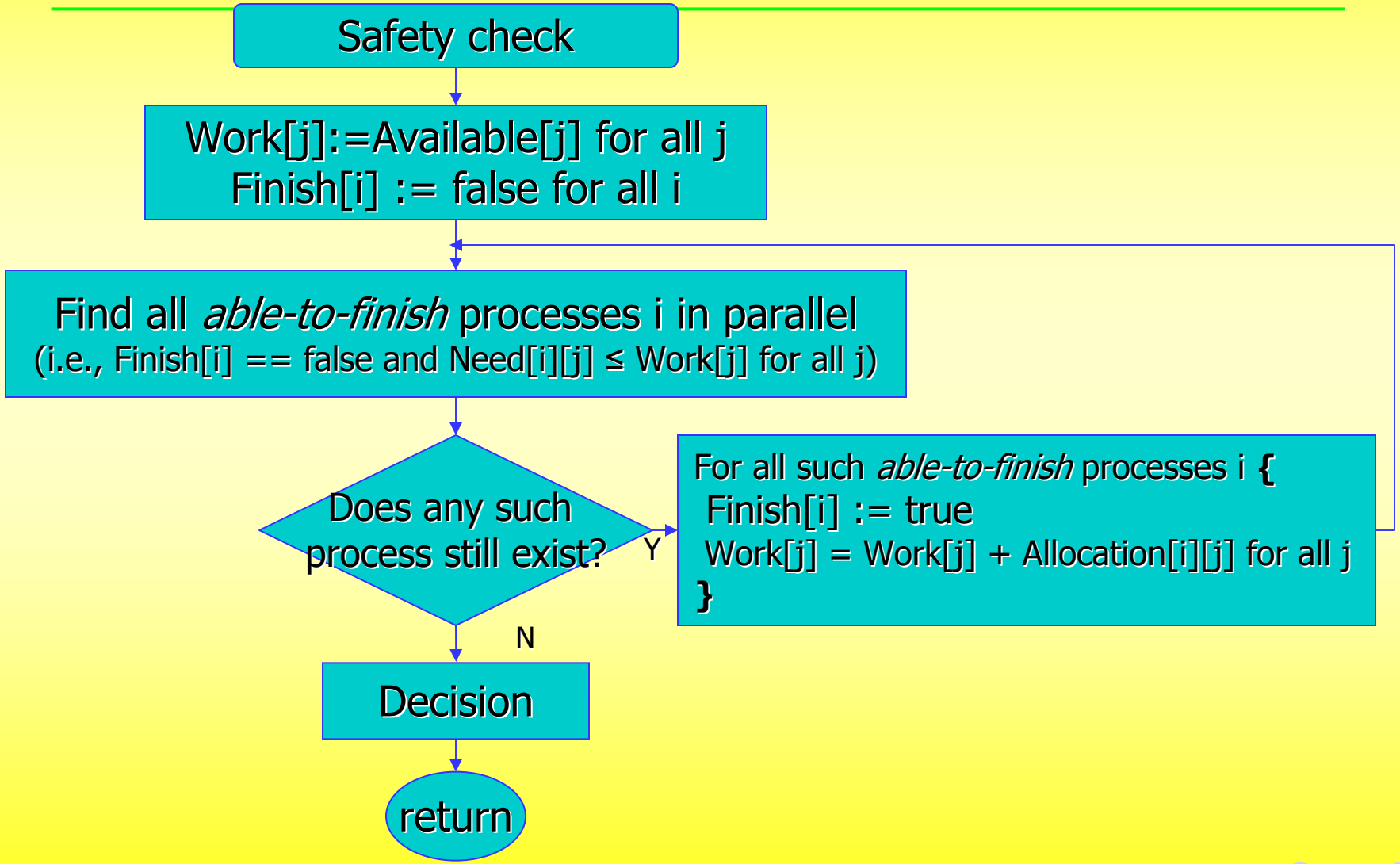
PBAU Architecture (3x3)



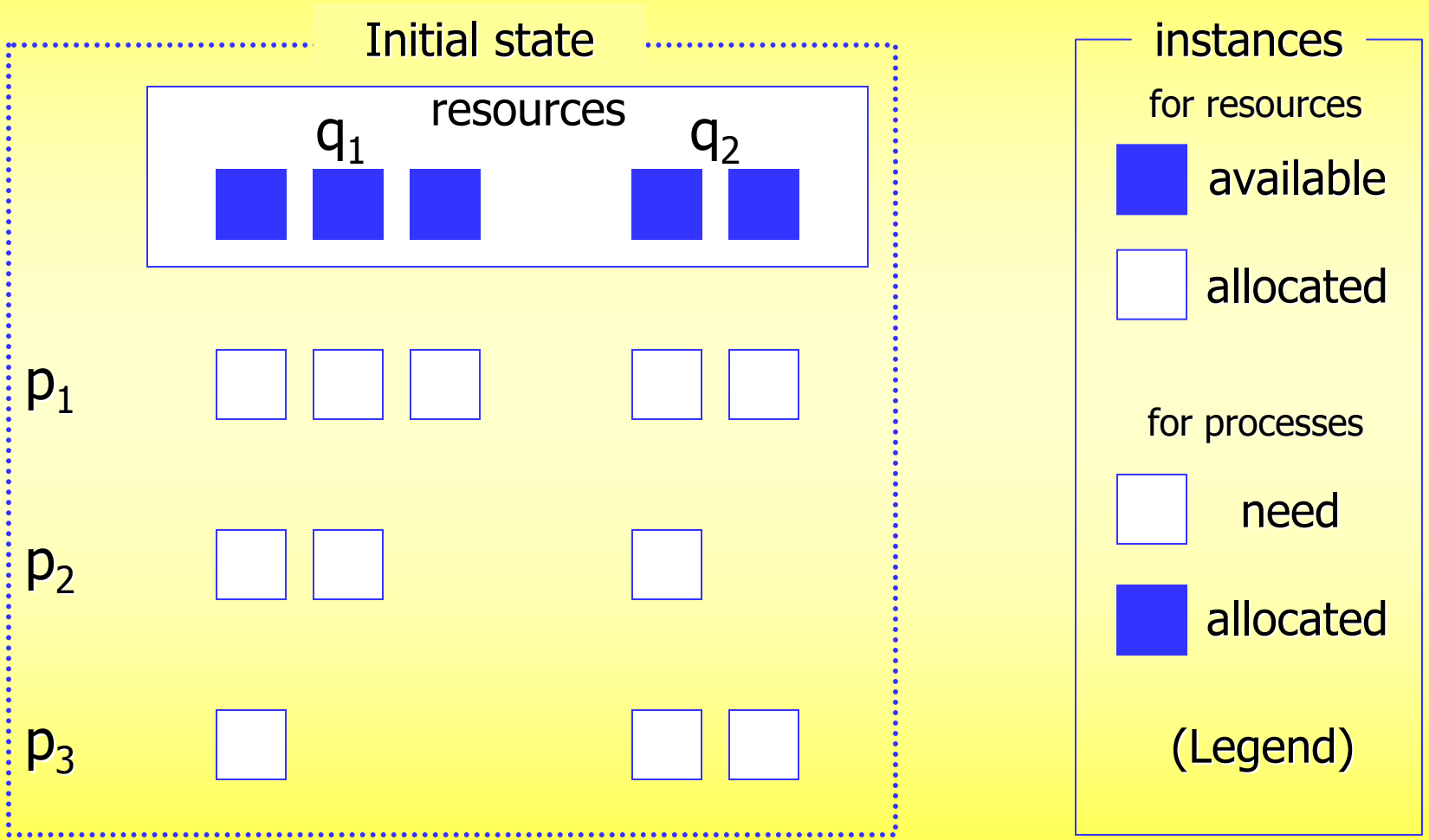
Parallel Banker's Algorithm (PBA)



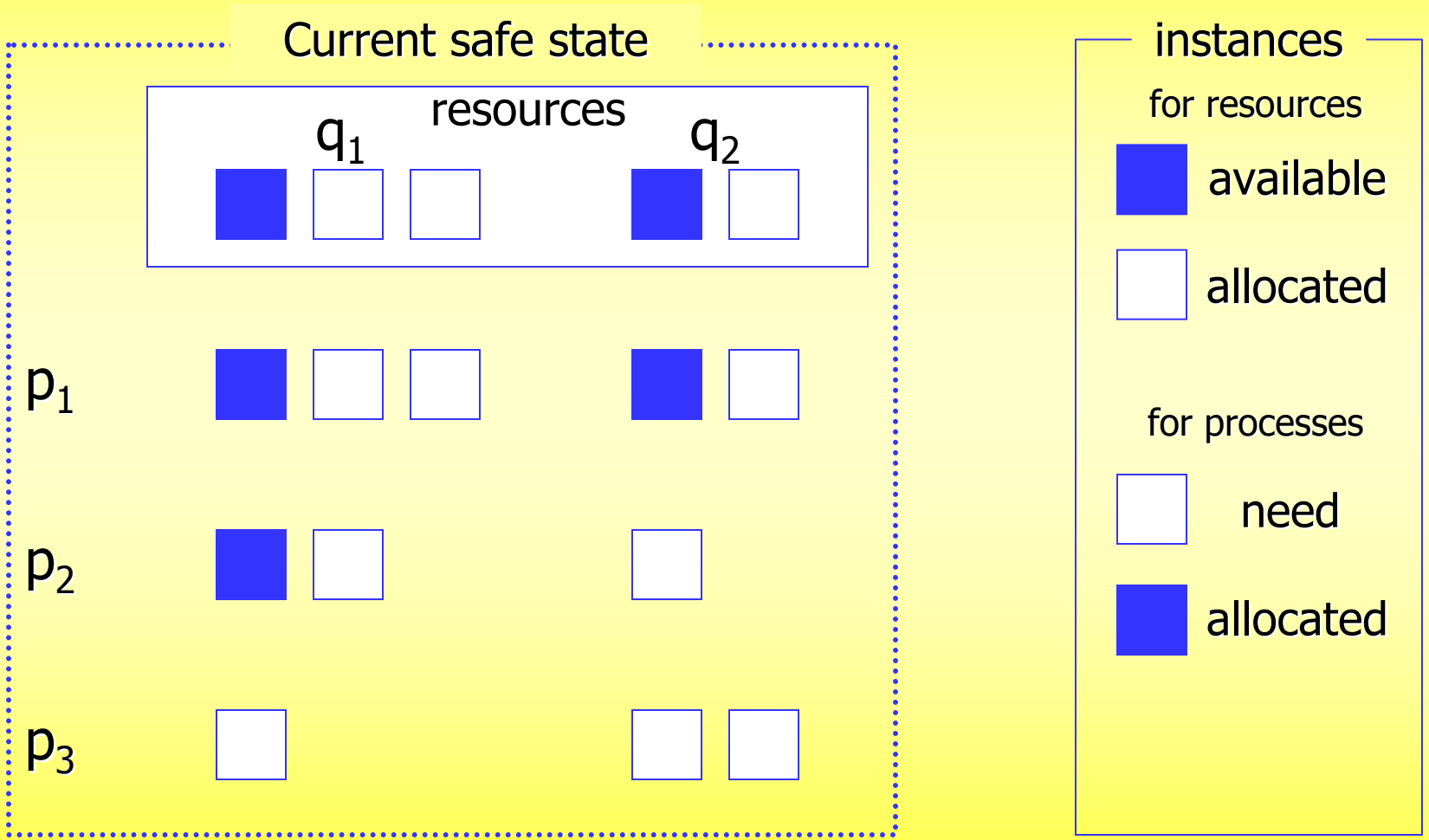
Parallel Banker's Algorithm (PBA)



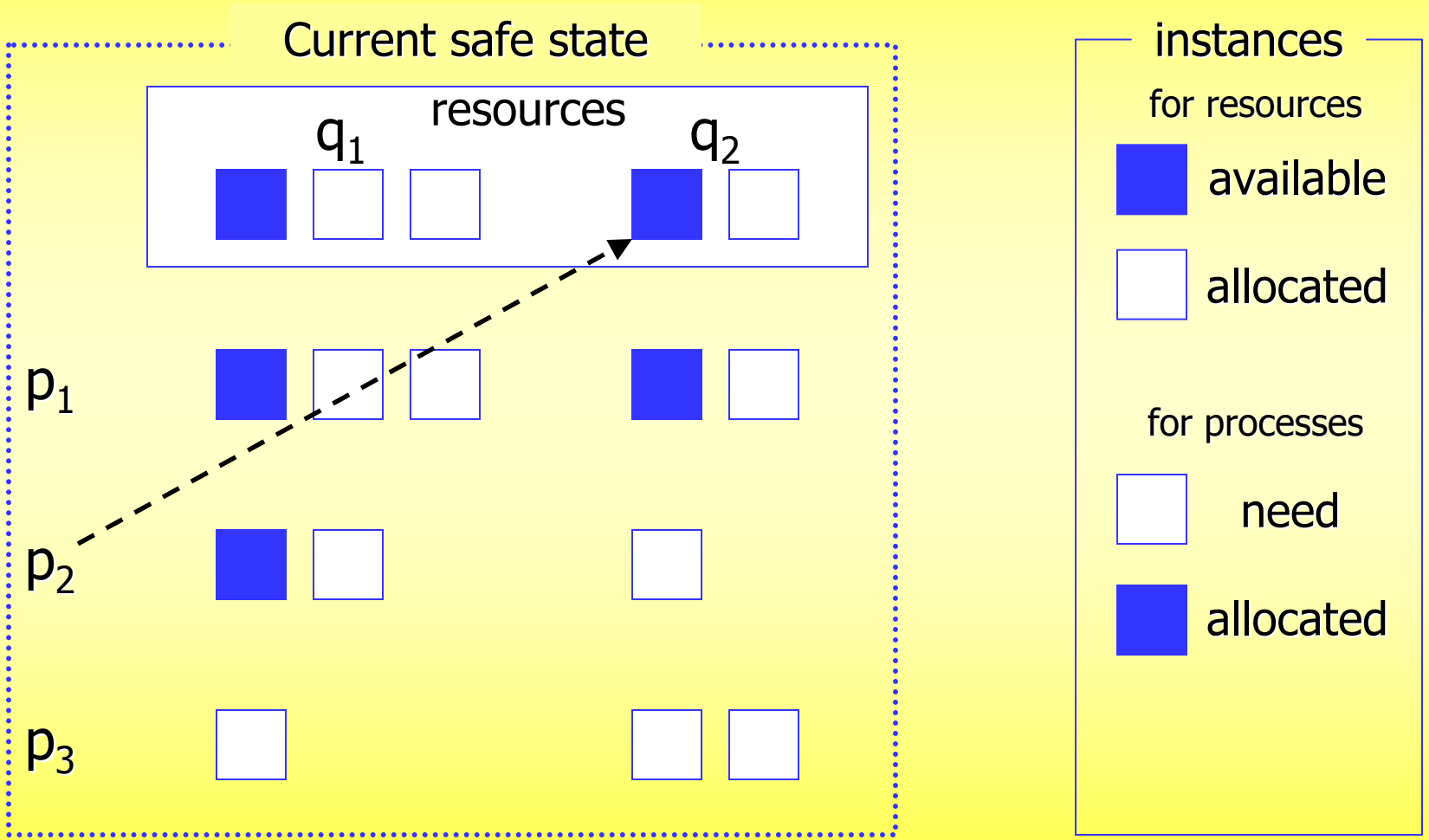
Resource Allocation by PBA



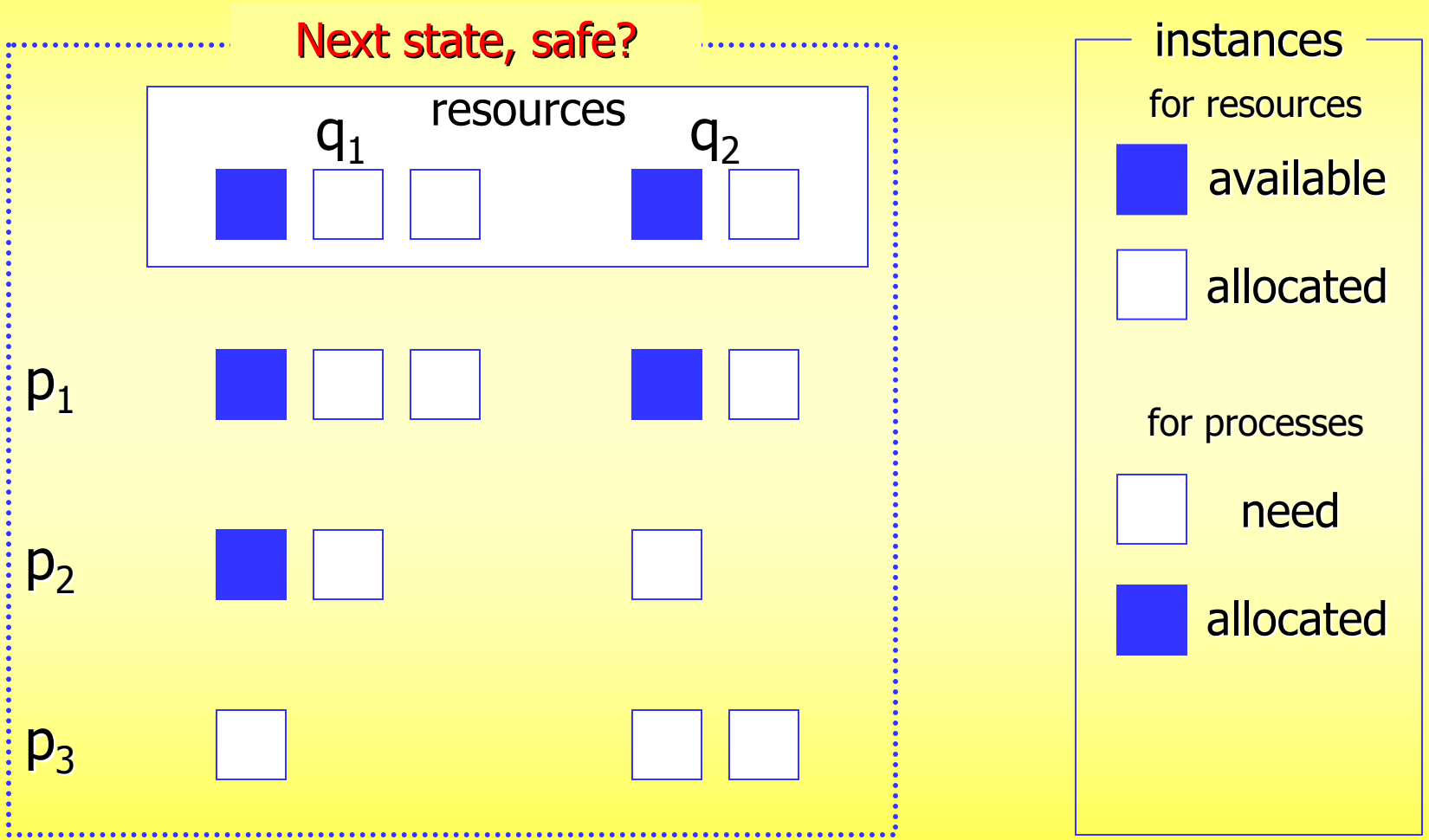
Resource Allocation by PBA



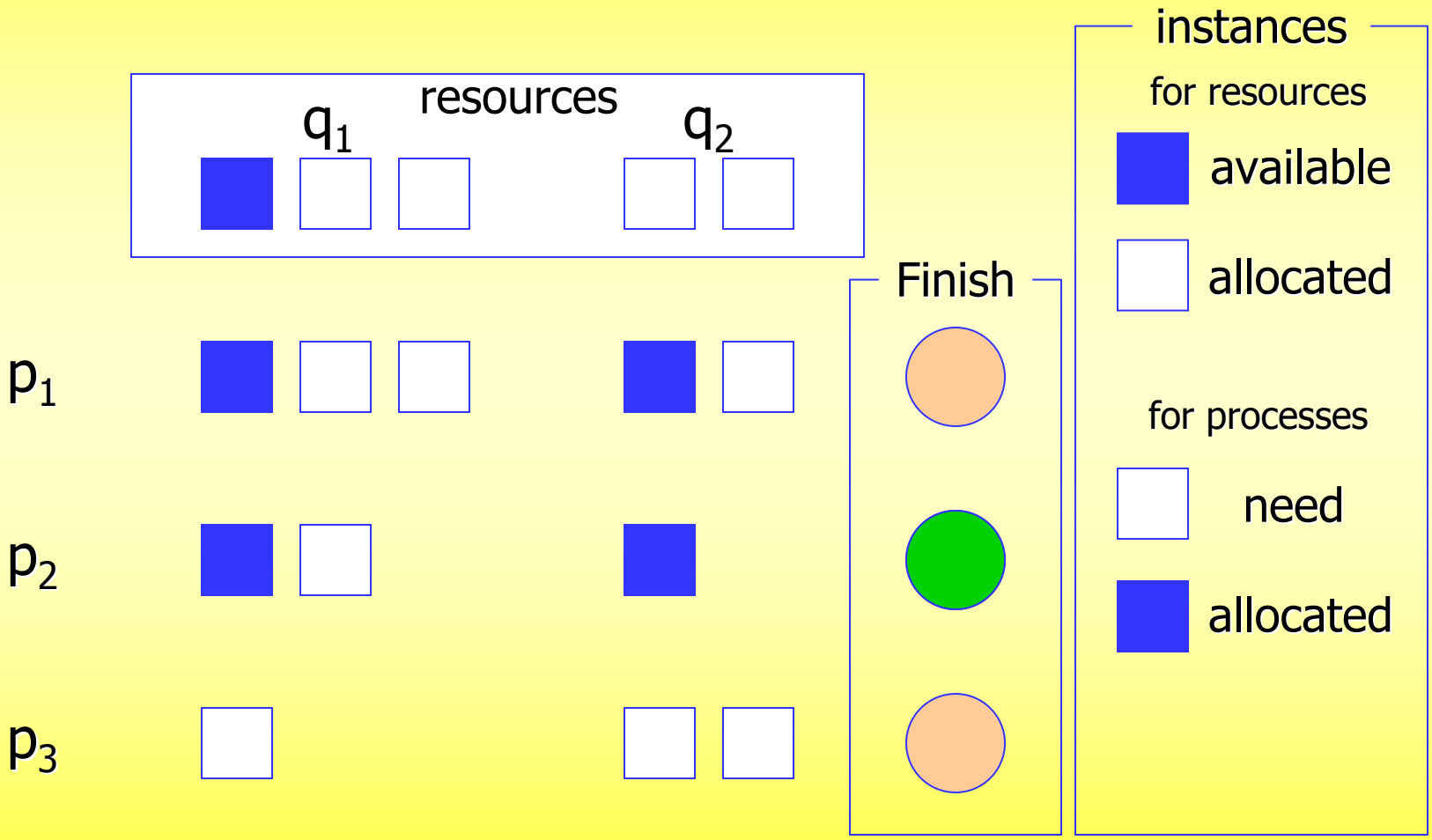
Resource Allocation by PBA



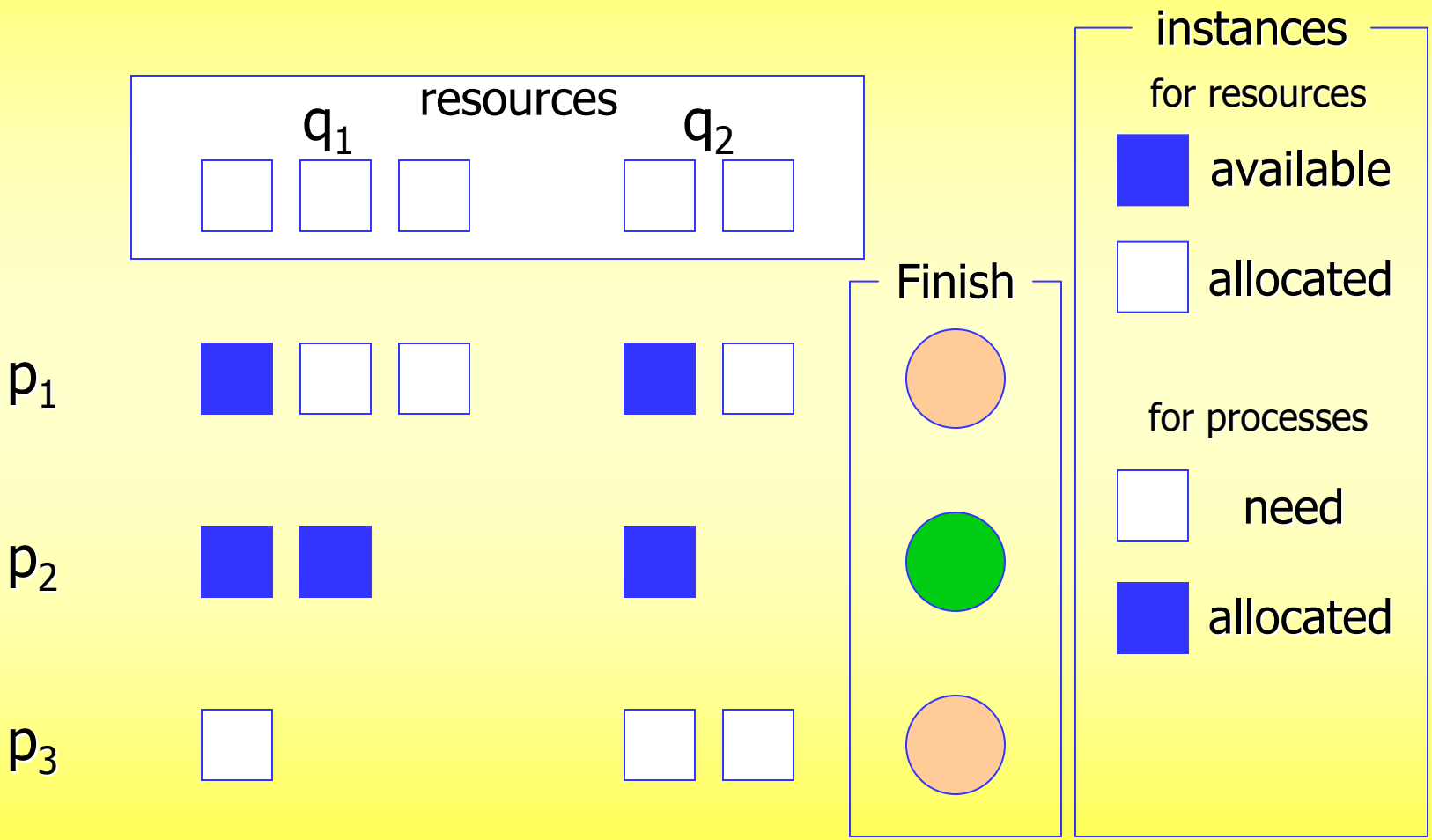
Resource Allocation by PBA



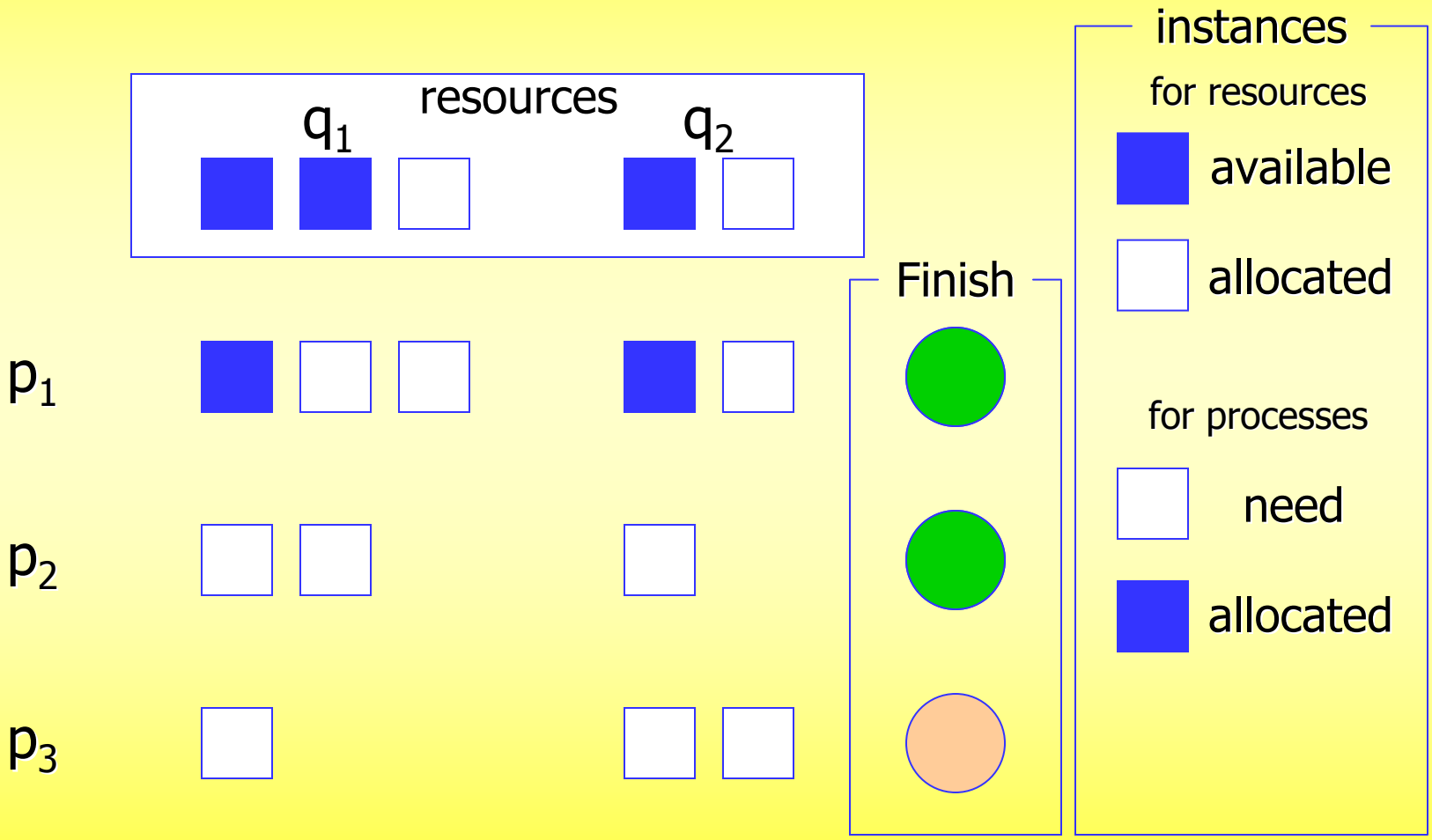
Resource Allocation by PBA



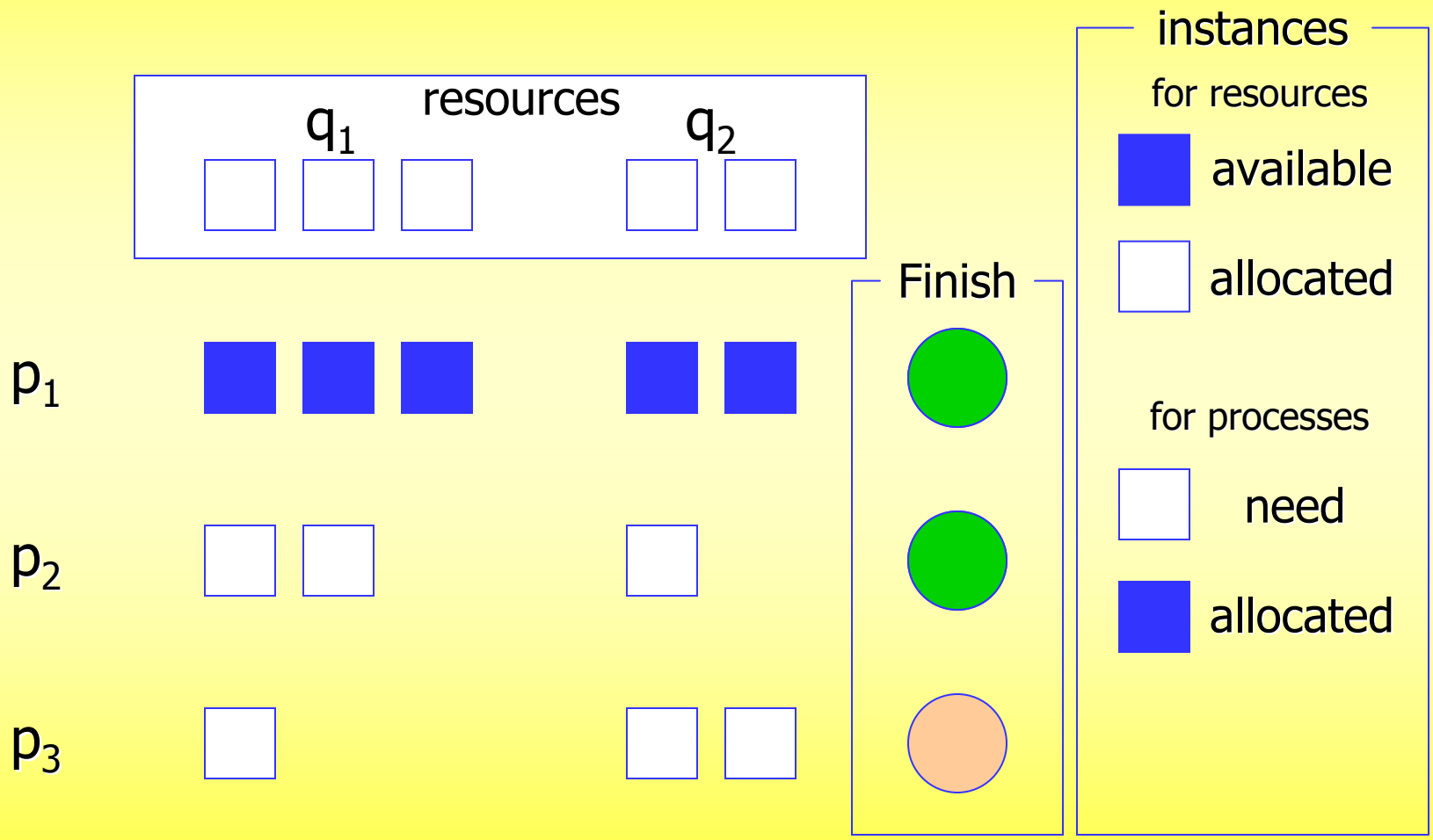
Resource Allocation by PBA



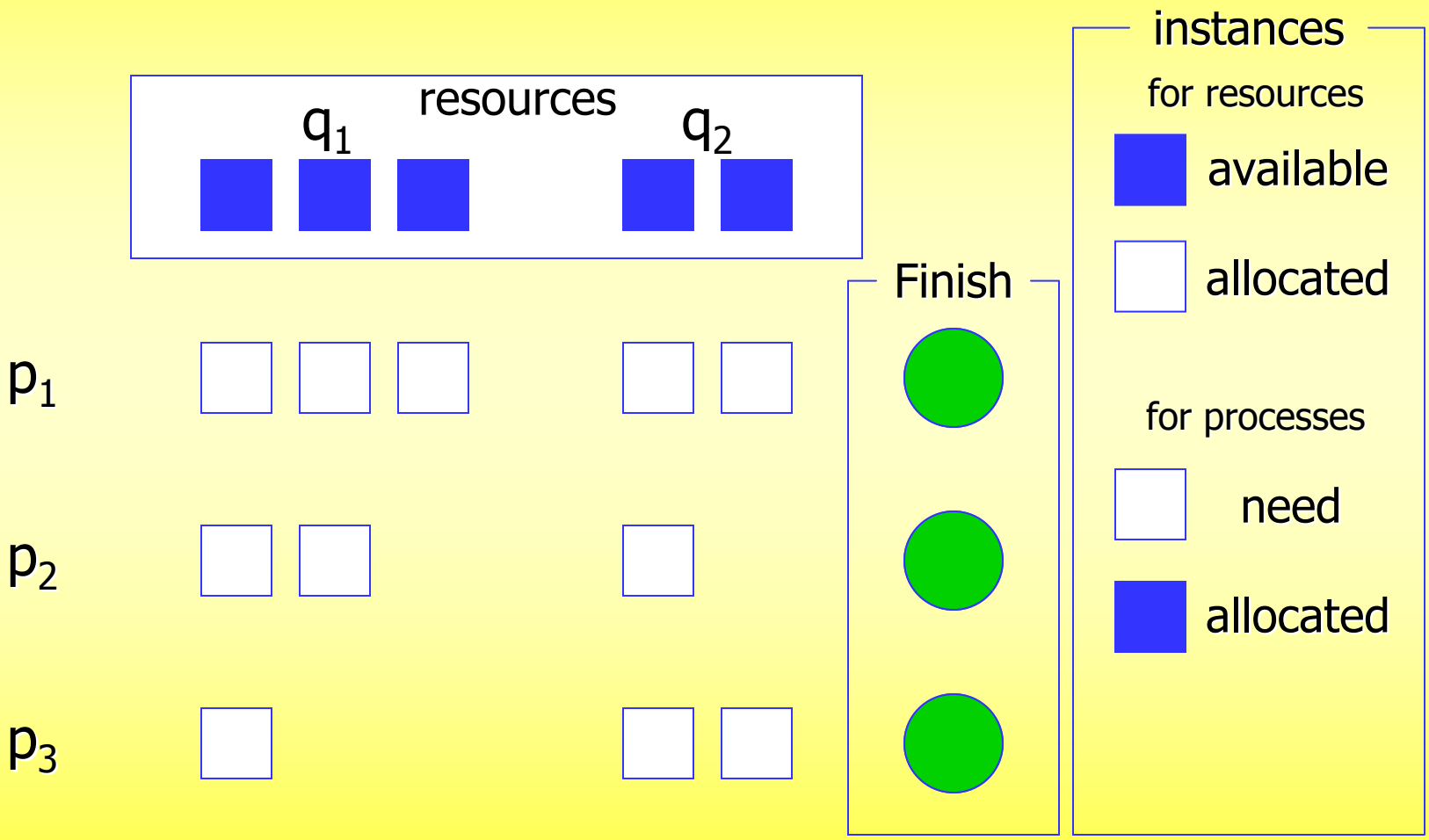
Resource Allocation by PBA



Resource Allocation by PBA

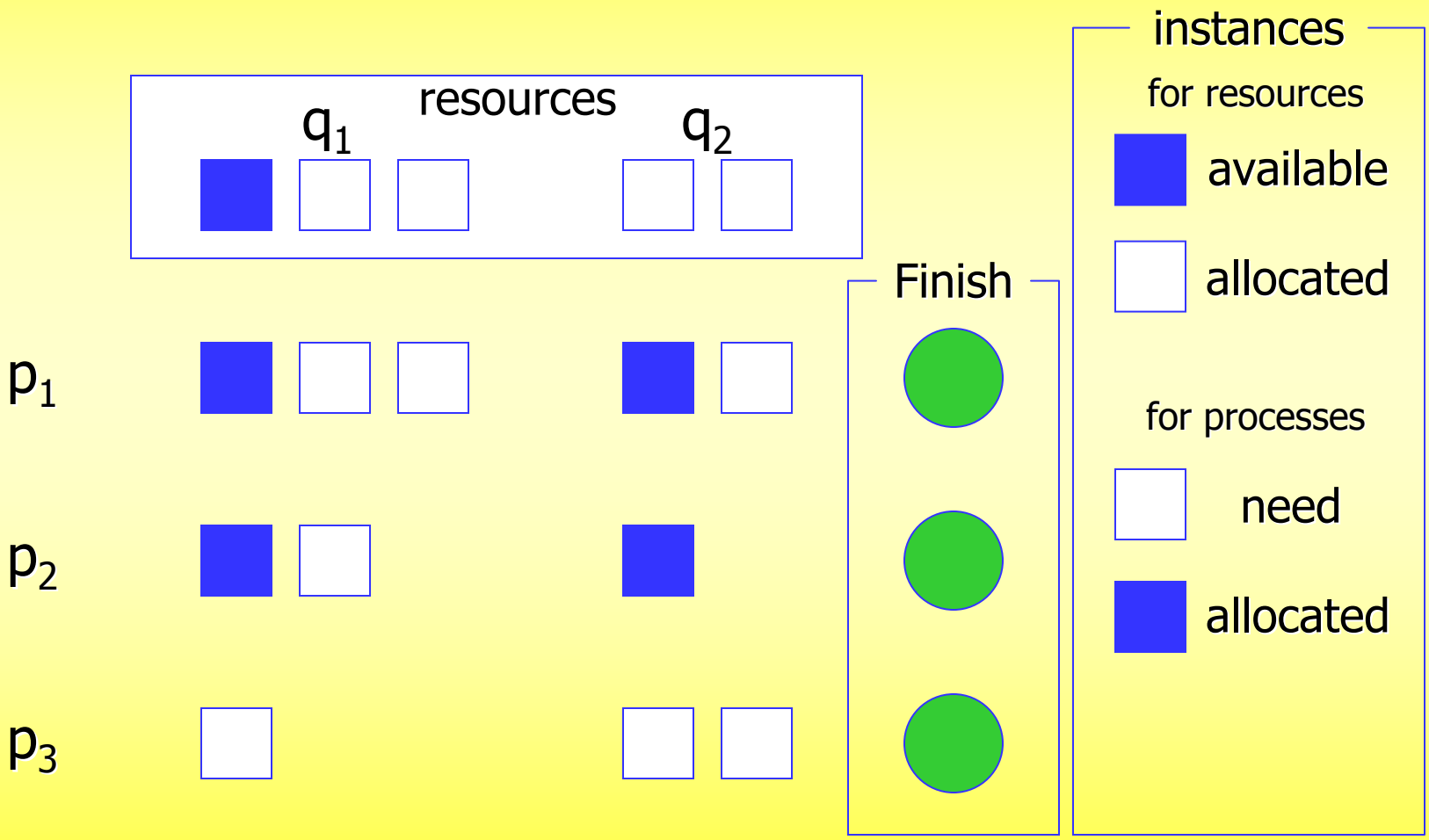


Resource Allocation by PBA



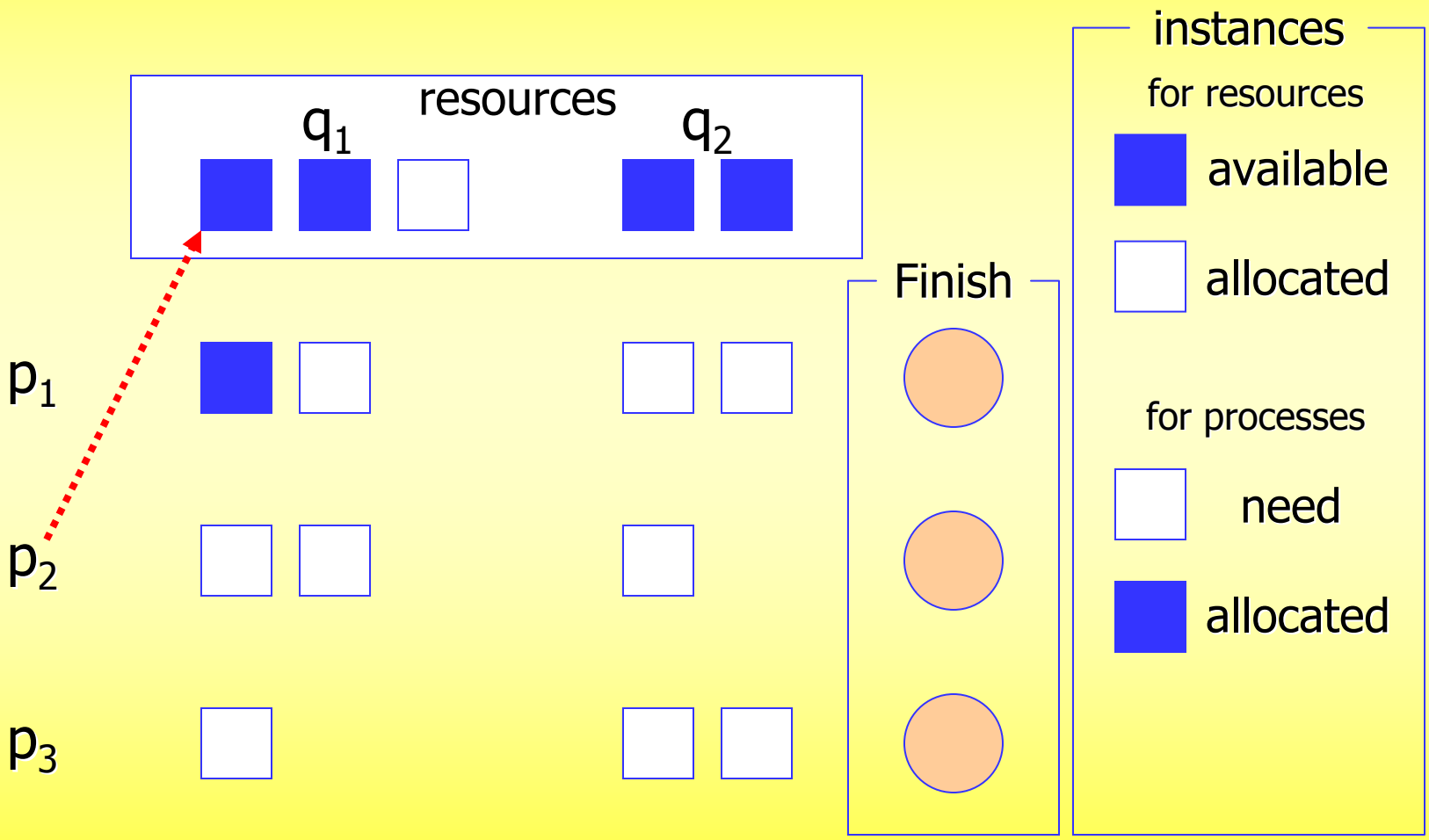
A safe sequence: $p_2 \rightarrow p_1 \rightarrow p_3$

Resource Allocation by PBA

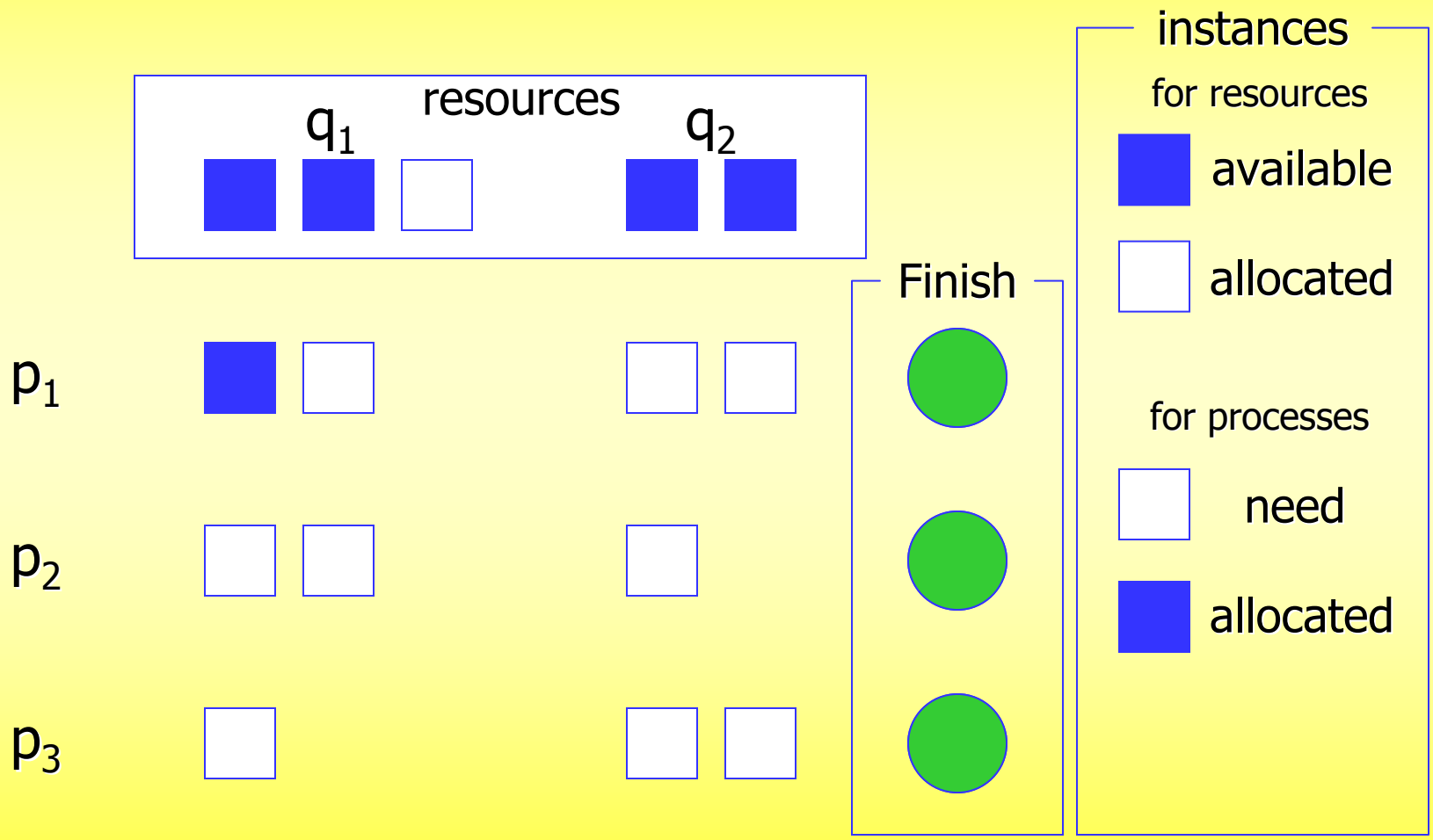


A safe sequence: $p_2 \rightarrow p_1 \rightarrow p_3$, 3 steps, i.e., O(n)

Resource Allocation by PBA
An example of O(1) run-time



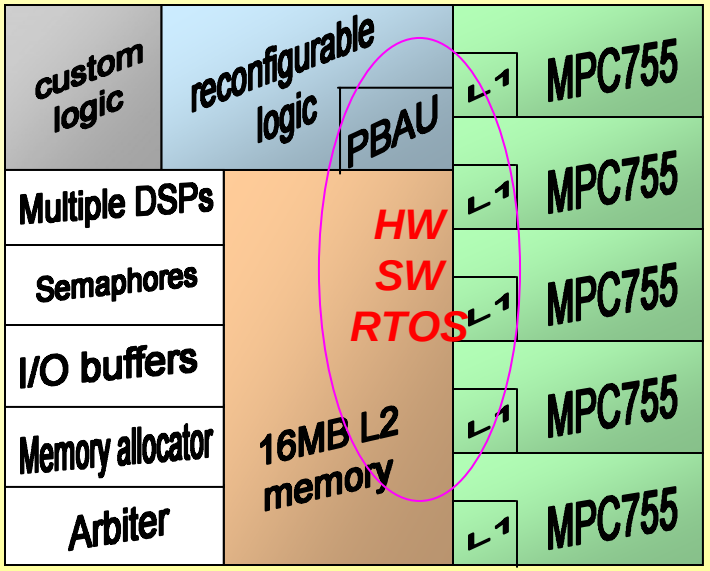
Resource Allocation by PBA
An example of O(1) run-time



A safe sequence: any combination

PBAU Experimentation - System and Application

- Five processors
- Four resources
  - Q1: Multiple DSPs
  - Q2: Hardware semaphores
  - Q3: I/O buffers
  - Q4: A memory allocator
- PBAU 5x5
- A robotic application
  - Five processes
  - Requires multiple instances
  - 22 times of service requests to PBAU
    - Requests, releases and claim settings



Georgia Tech

Experimental Results of PBAU

- Performance improvement
  - 99% algorithm execution time reduction
  - 19% reduction in an application execution time

Method of Deadlock Avoidance	Algorithm Exe. Time (cycles)	Normalized Exe. Time	Application Exe. Time (cycles)
PBAU Hardware	3.32 (average)	1X	185716
BA in Software	5398 (average)	1625X	221259

Synthesis Results of PBAU

- Synopsys Design Compiler
- TSMC .25µm technology library from Qualcore Logic
- 0.05% of the total SoC area with five PEs and memory
- All PBAUs able to handle up to 16 instances for each resource

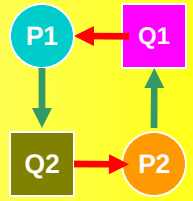
Module Name	Total Area in terms of two-input NAND gates	Lines of Verilog HDL Code
PBAU 5x5	1303	600
8x8	3243	700
10x10	5030	770
15x15	11158	1000
20x20	19753	1350
MPSoC	42 Million	-

Clock period used: 4 ns

TSMC: Taiwan Semiconductor Manufacturing Company

PE: Processing Element

Outline



- Motivation and Objective
- Background and Prior Work
- Proofs about Deadlock Detection Unit
- Deadlock Avoidance Unit
- Parallel Banker's Algorithm Unit
- Integration into the δ hardware/software RTOS framework
- Conclusion

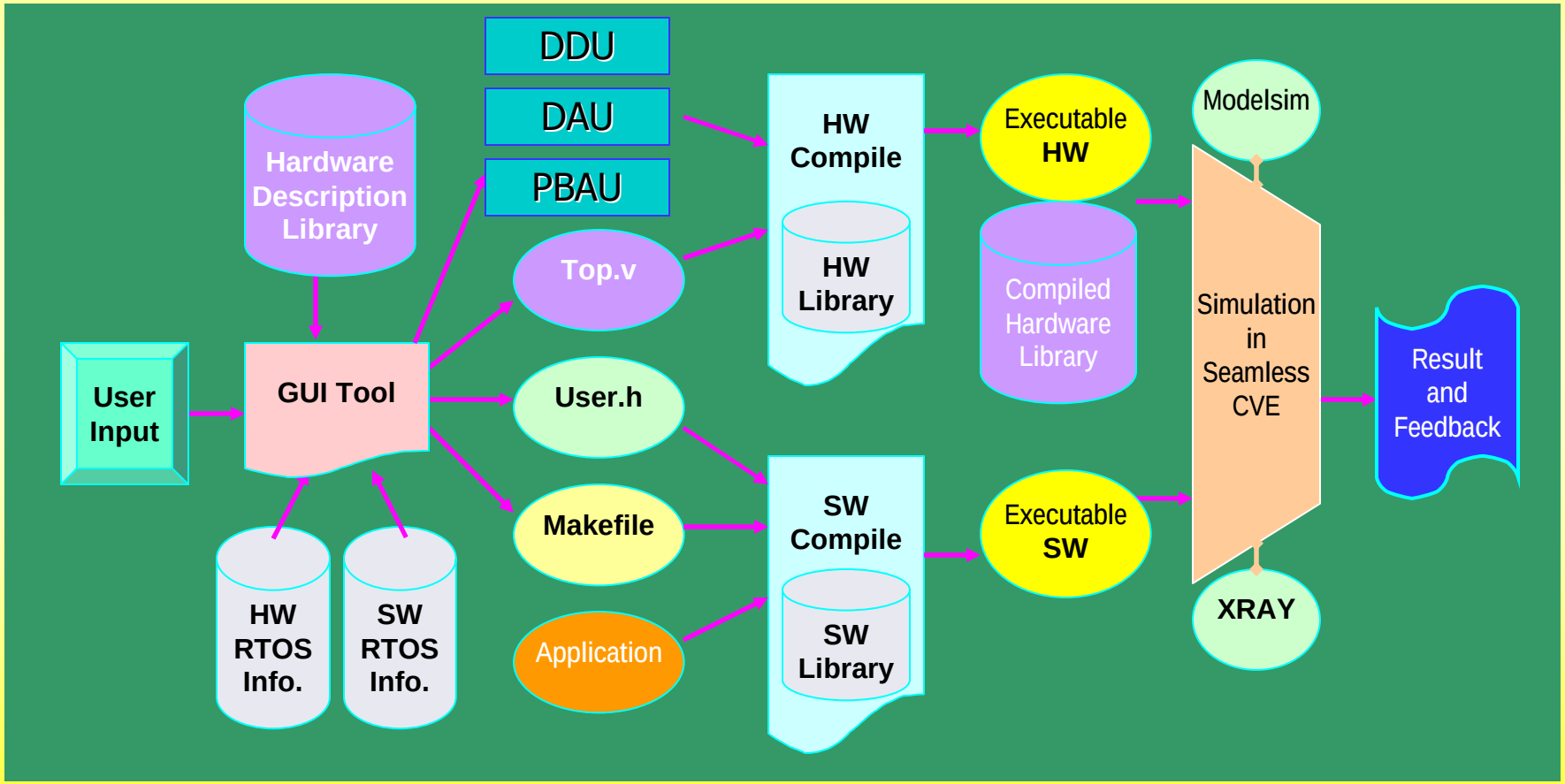
Georgia Tech

δ framework

- Hardware/Software RTOS/MPSoC Configuration Framework
  - Enables automatic generation of different mixes of the HW/SW RTOS
  - Can be generalized to instantiate additional HW or SW RTOS components
  - Integrates parameterized IP Generators such as DDU, DAU and PBAU generators
- Designed by Mooney and Lee
- RTOS Components: designed by B. Akgul, P. Kuarchroen, J. Lee, K. Ryu, M. Shalan and E. Shin

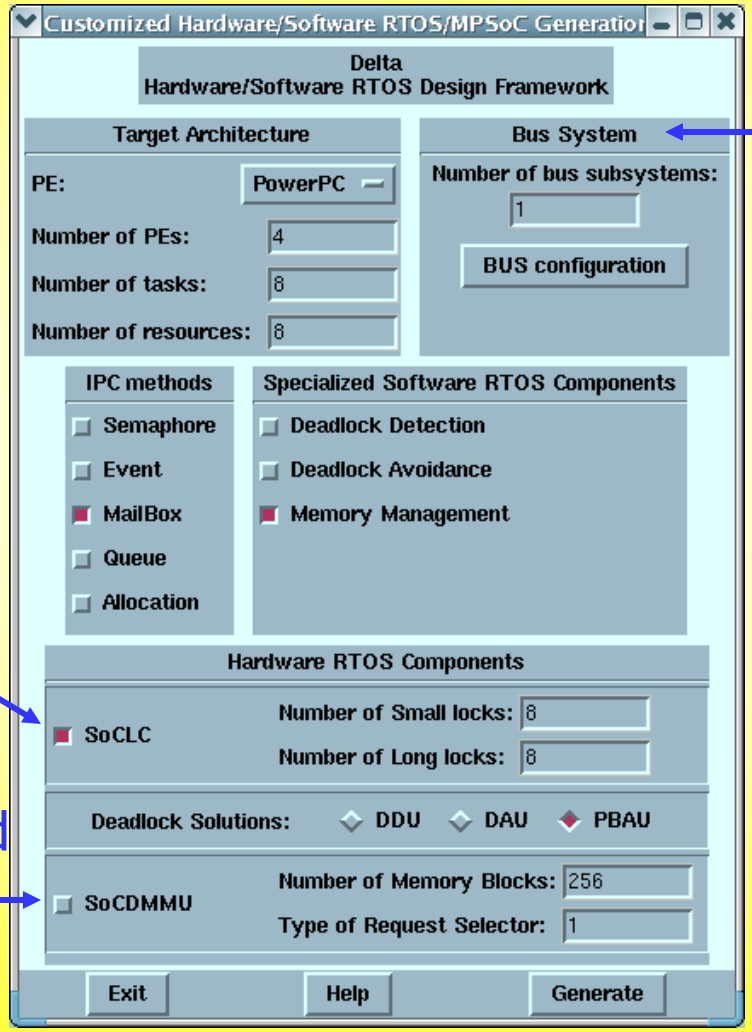
δ framework
- Methodology

- δ Framework



δ framework
- Methodology

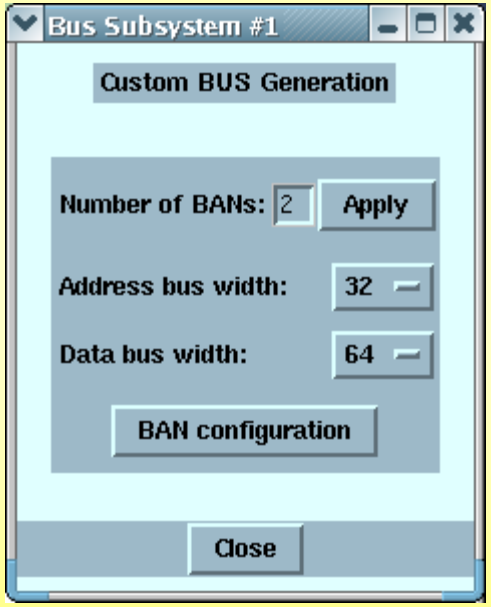
- GUI
- SoCLC: designed by B.Akgul
- SoCDMMU: designed by M. Shalan
- Bus: designed by K. Ryu

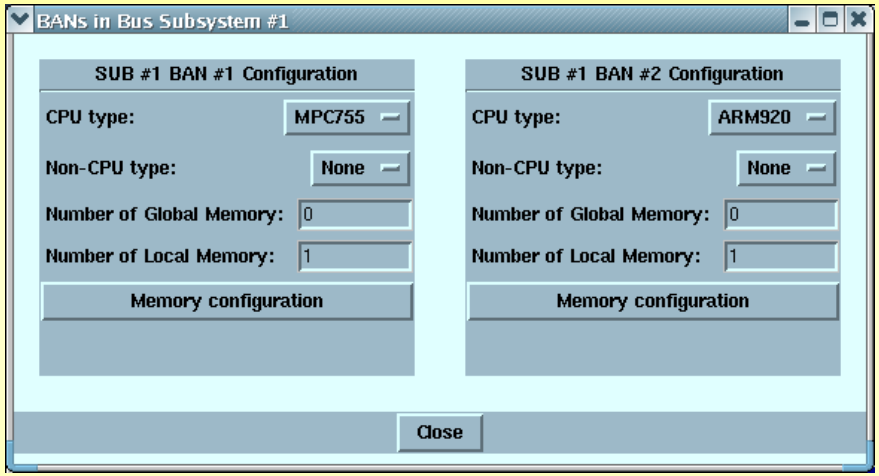


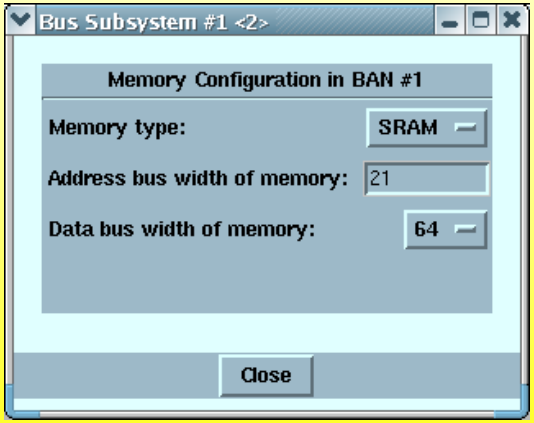
Georgia Tech

δ framework
- Methodology

- GUIs for a custom bus generation



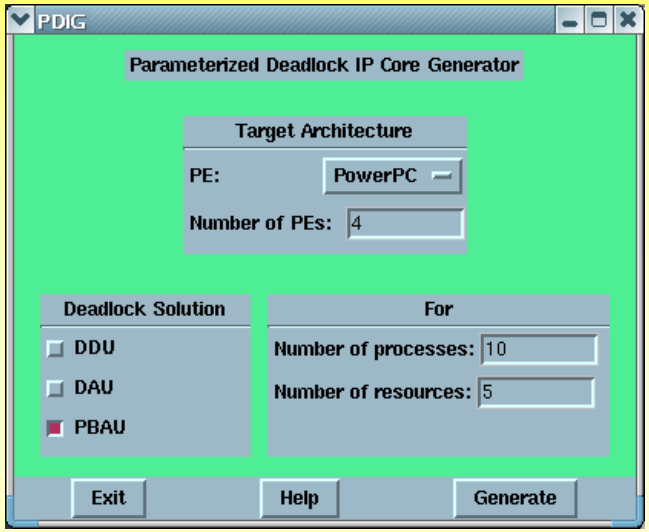


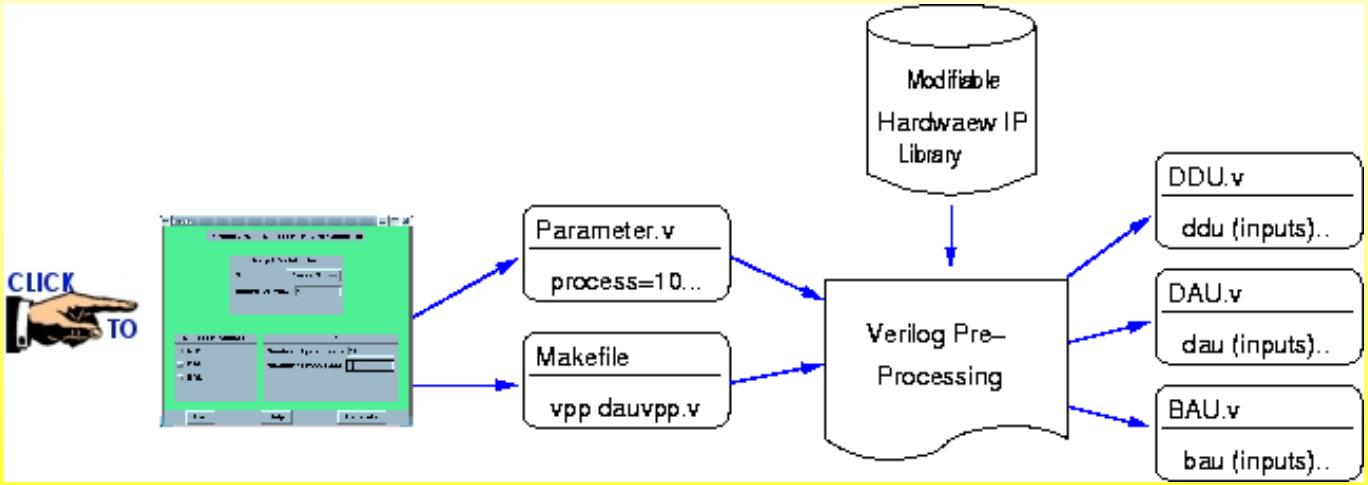


Georgia Tech

Deadlock IP generator

- GUI

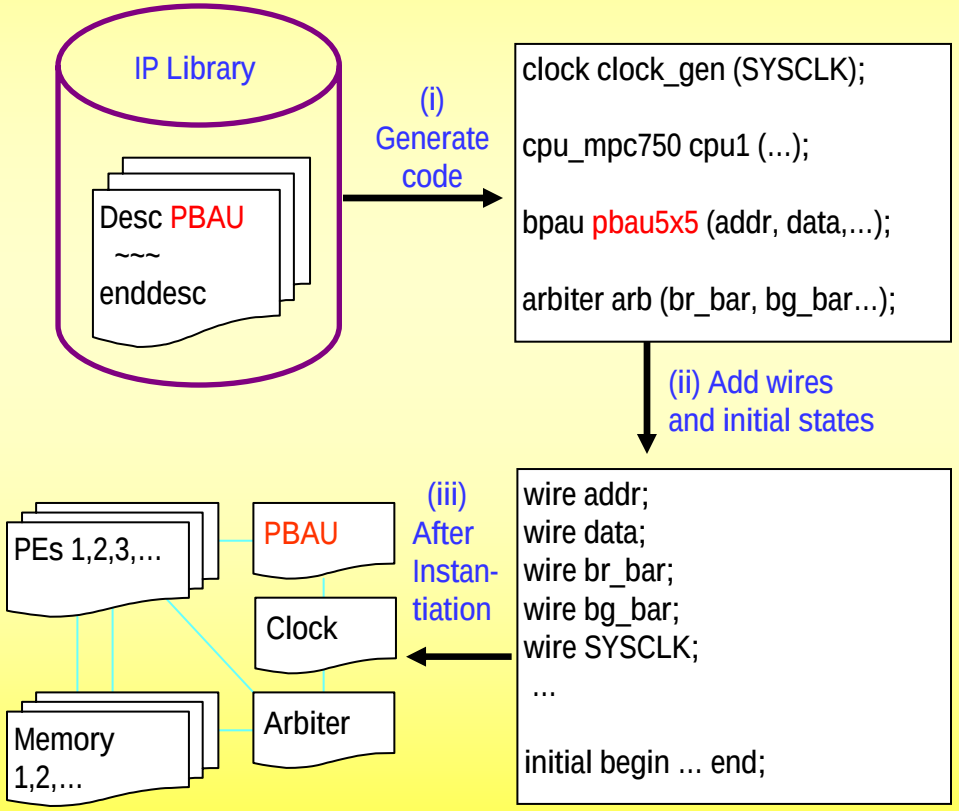




Georgia Tech

δ framework - Implementation

- Verilog top file generation example
  - Start with PBAU description
  - Generate instantiation code
    - ✓ multiple instantiation code if needed (e.g., PEs)
  - Add wires and initial statements



Georgia Tech

Conclusion with Contribution

- Proofs of Deadlock Detection Unit (DDU)
  - Correctness and run-time complexity
- Deadlock Avoidance Unit (DAU)
  - Faster Deadlock Avoidance (312X)
    - No prior knowledge about resource requirements
    - No restrictions on resource usage
    - Higher resource utilization
    - Solution to livelock
- Parallel Banker's Algorithm Unit (PBAU)
  - Faster deadlock avoidance for multiple instance multiple resource systems (1600X)
  - Small area (less than 0.1% in our example SoC)
- δ Hardware/Software RTOS partitioning framework
  - With custom deadlock IP generator for a specific target

Publications

[1] J. Lee and V. Mooney "An O(n) Parallel Banker's Algorithm for System-on-a-Chip," submitted to *Design, Automation and Test in Europe (DATE'05),* under review.

[2] J. Lee and V. Mooney "Hardware/Software Partitioning of Operating Systems: Focus on Deadlock Detection and Avoidance," to be appeared in *IEE Computer & Digital Techniques (IEE CDT),* January 2005.

[3] J. Lee and V. Mooney "An O(min(m,n)) Parallel Deadlock Detection Algorithm," resubmitted to *ACM Transactions on Design Automation of Electronic Systems (TODAES)* on September 2004, under review.

[4] J. Lee and V. Mooney "A Novel Deadlock Avoidance Algorithm and Its Hardware Implementation," International Conference on Hardware/Software Codesign and System Synthesis *(CODES'04)*, pp. 200-205, September 2004.

[5] J. Lee, V. Mooney, A. Daleby, K. Ingstrom, T. Klevin and L. Lindh, "A comparison of the RTU hardware RTOS with a Hardware/Software RTOS," *Proceedings of the Asia South Pacific Design Automation Conference (ASPDAC 2003),* pp. 683-688, January 2003. Best Paper Award Candidate (one of 12 nominees; not selected for Best Paper).

[6] J. Lee, K. Ryu and V. Mooney, "A framework for automatic generation of configuration files for a custom RTOS," *Proceedings of the Engineering of Reconfigurable Systems and Algorithms (ERSA 2002),* pp. 31-37, June 2002.

[7] J. Lee and V. Mooney "An O(min(m,n)) Parallel Deadlock Detection Algorithm," Tech. Rep. GIT-CC-03-41, College of Computing, Georgia Institute of Technology, Atlanta, GA. September 2003.

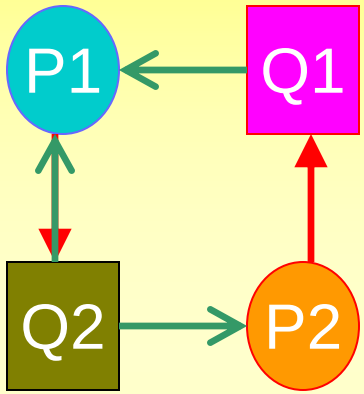
[8] B. Akgul, J. Lee and V. Mooney, "A System-on-a-Chip Lock Cache with Task Preemption Support," *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES 2001),* pp. 149-157, November 2001.

Q & A

Thank you!

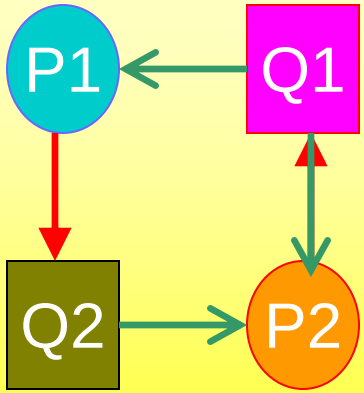
More about R-dl avoidance

If P1>P2



Ask P2 to release Q2
Q2 can be granted to P1

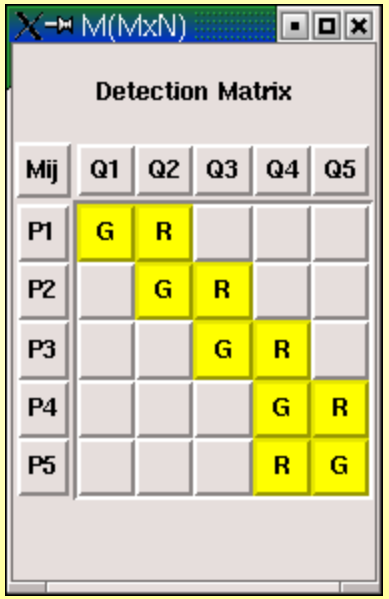
If P2>P1



Ask P1 to release Q1
Q1 can be granted to P2

Proof of the Correctness of DDU Algorithm

- Correctness
  - Not empty $\Rightarrow \exists$ cycle(s) $\Rightarrow$ deadlock
  - Empty $\Rightarrow \exists$ no cycle(s) $\Rightarrow$ no deadlock
- Proof with
  - 5 Lemmas
  - 4 Theorems



* Additional detail can be found in a journal submission and a technical report [3, 7]

Proof of the Correctness of DDU Algorithm (cont'd)

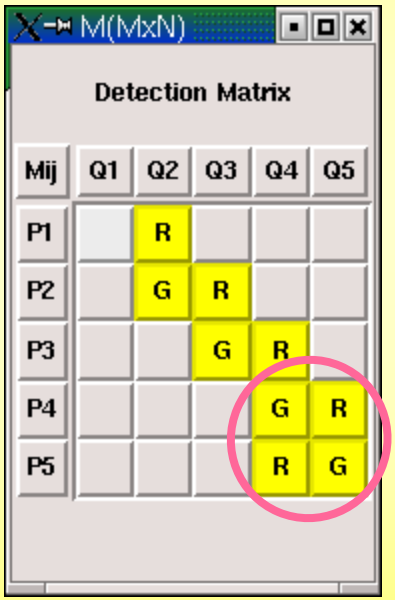
- 5 Lemmas
  - Removing terminal edges will not alter any cycle
  - If a RAG can be completely reduced, the system does not have a deadlock
  - If no cycles in a RAG, the RAG can be completely reduced.
  - A process that is making progress is not involved in deadlock
  - If a system does not have a deadlock, all processes can make progress within a finite time

Proof of the Correctness of DDU Algorithm (cont'd)

- 4 Theorems
  - If a RAG contains any cycle, the RAG cannot be completely reduced
  - If a RAG cannot be completely reduced, the RAG contains at least a cycle
  - A cycle is a necessary and sufficient condition for deadlock
  - DDU Algorithm detects deadlock iff there exists a cycle in a RAG

Proof of Run-Time Complexity of DDU

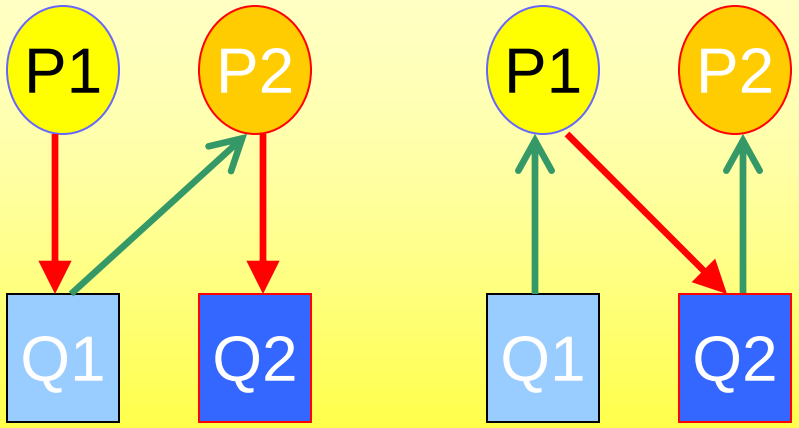
- Q: How many steps does DDU need to detect deadlock in parallel hardware?
- Proof with
  - 2 Corollaries
  - 1 Lemma
  - 1 Theorem



* Additional detail can be found in a journal submission and a technical report [3, 7]

Proof of Run-Time Complexity of DDU (cont'd)

- 2 Corollaries
  - The total number of nodes in the smallest possible cycle = 4
  - The number of edges in any path (not cycle) using all nodes in the smallest possible cycle = 3



Comparison: DDU and DAU

Method of Deadlock Detection	Detection Time (average cycles)	Normalized Detection Time
DDU	1.3	1X
DDU in software	1830	1408X

Method of Deadlock Avoidance	Avoidance Time (average cycles)	Normalized Avoidance Time
Hardware	7	1X
Software	2188	312X

