A Lattice-Based Framework for the Classification and Design of Asynchronous Pipelines

Peggy B. McGee and Steven M. Nowick

Department of Computer Science
Columbia University

Why Study Asynchronous Pipelines

- No global clock
  - avoids global timing issues
- Adaptability to environments running at different speeds
  - mixed sync / async interfaces
  - different clock rates in left and right environments
  - dynamically-varying clock speed (due to voltage scaling, etc.)
- Reusability
- Provides very high speed: "gate-level" pipelining (multi-GHz)

Overview

Many pipeline protocols have been proposed.

Overview

Many pipeline protocols have been proposed.

Examples: pipelines using dynamic logic with no latches

PCHB [Lines98]

HC [Singh/Nowick00]

PCFB [Lines98]

PS0 [Williams91]

LP3/1 [Singh/Nowick00]

PC0 [Williams91]

Overview

Many pipeline protocols have been proposed.

Examples: pipelines using dynamic logic with no latches

PCHB [Lines98]

HC [Singh/Nowick00]

PCFB [Lines98]

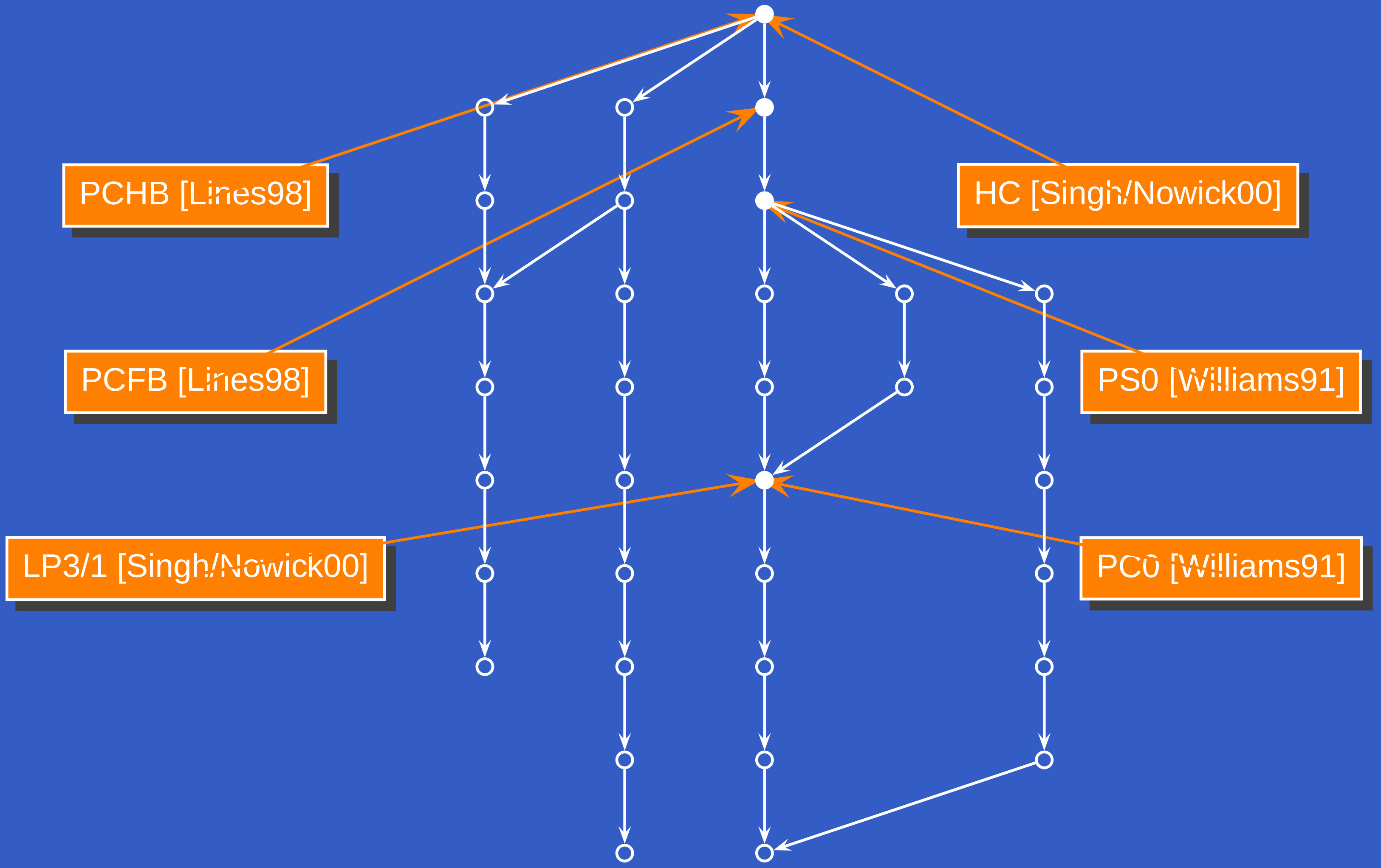
PS0 [Williams91]

LP3/1 [Singh/Nowick00]

PC0 [Williams91]

Putting them together in a unifying framework:

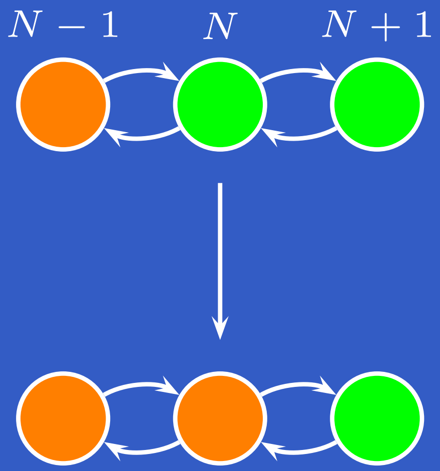
Overview


PCHB [Lines98]
HC [Singh/Nowick00]
PCFB [Lines98]
PS0 [Williams91]
LP3/1 [Singh/Nowick00]
PC0 [Williams91]

Asynchronous Pipelines: Protocols

Basic protocol: stage N synchronizes with its 2 neighbors

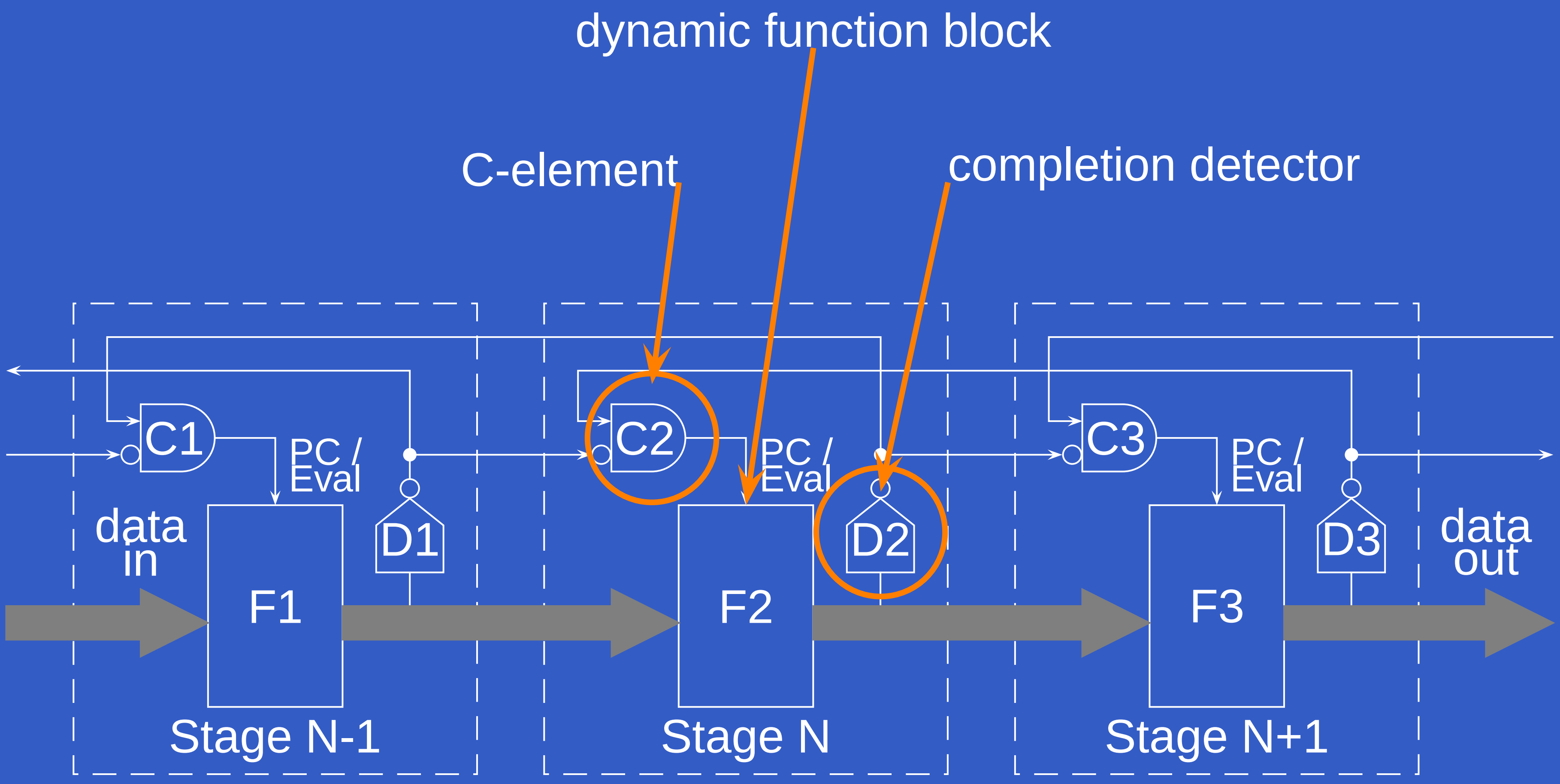
- When new data is available from stage N-1
- And stage N+1 has consumed current result
- Stage N computes a new value, sends it to its stage N+1, and sends acknowledgment to stage N-1



Many variants

- dual-rail data encoding vs. single-rail bundled data
- two-phase vs. four-phase handshake signaling
- dynamic logic vs. static logic
- linear structures vs. ring structures
- timing robust vs. using timing assumptions

Asynchronous Pipeline: PC0

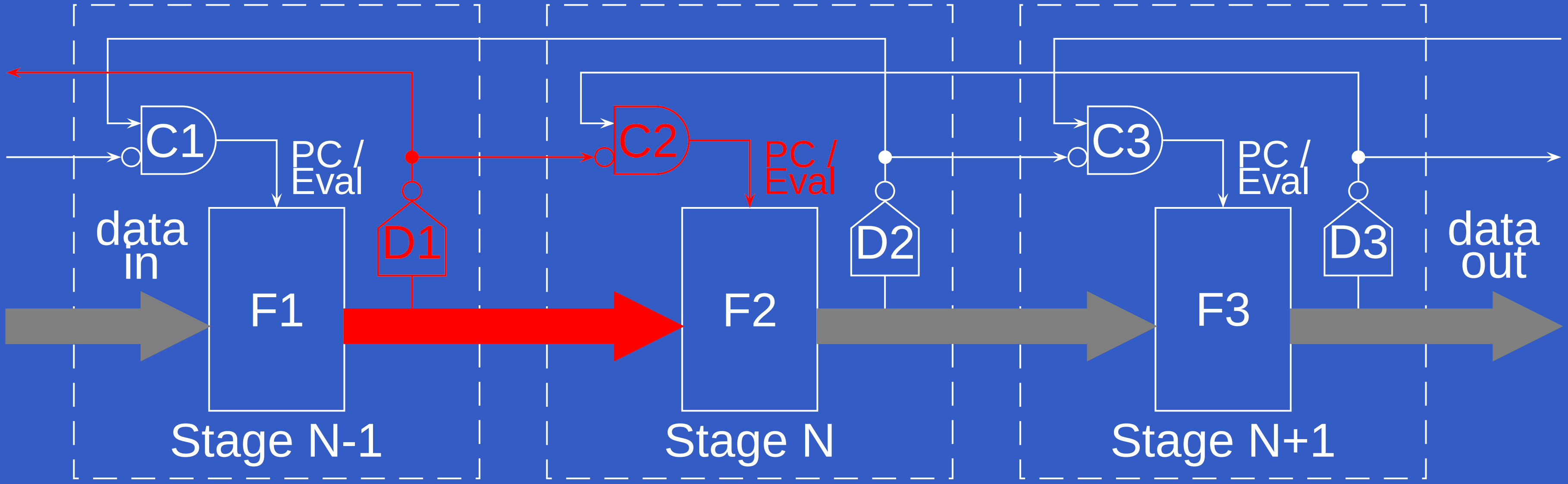


PC0, Williams[1991]

Asynchronous Pipeline: PC0

Stage N is enabled to evaluate when:

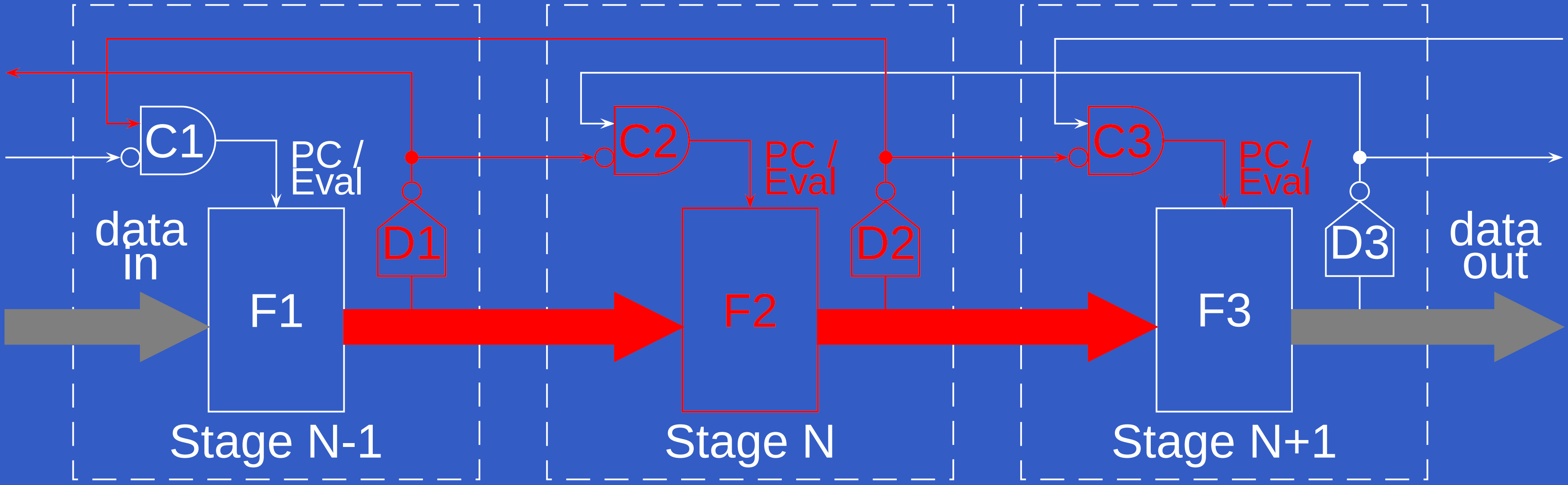
- stage N-1 has new data
- stage N+1 has completed reset of previous data token



PC0, Williams[1991]

Asynchronous Pipeline: PC0

Stage N evaluates

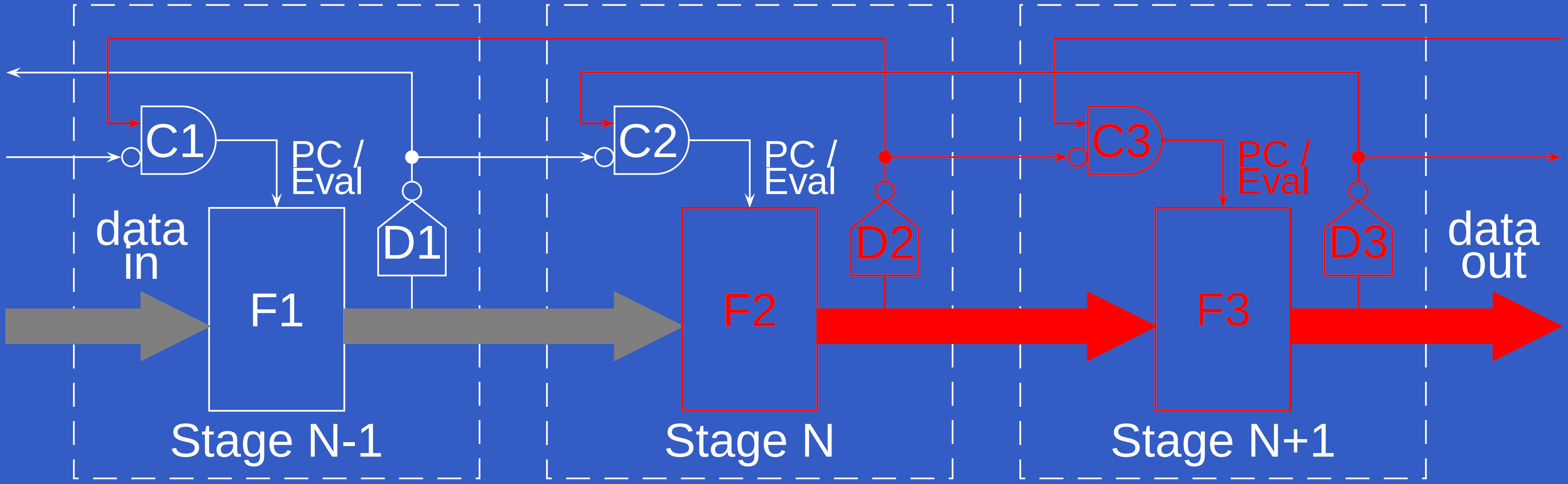


PC0, Williams[1991]

Asynchronous Pipeline: PC0

Stage N is enabled to precharge when

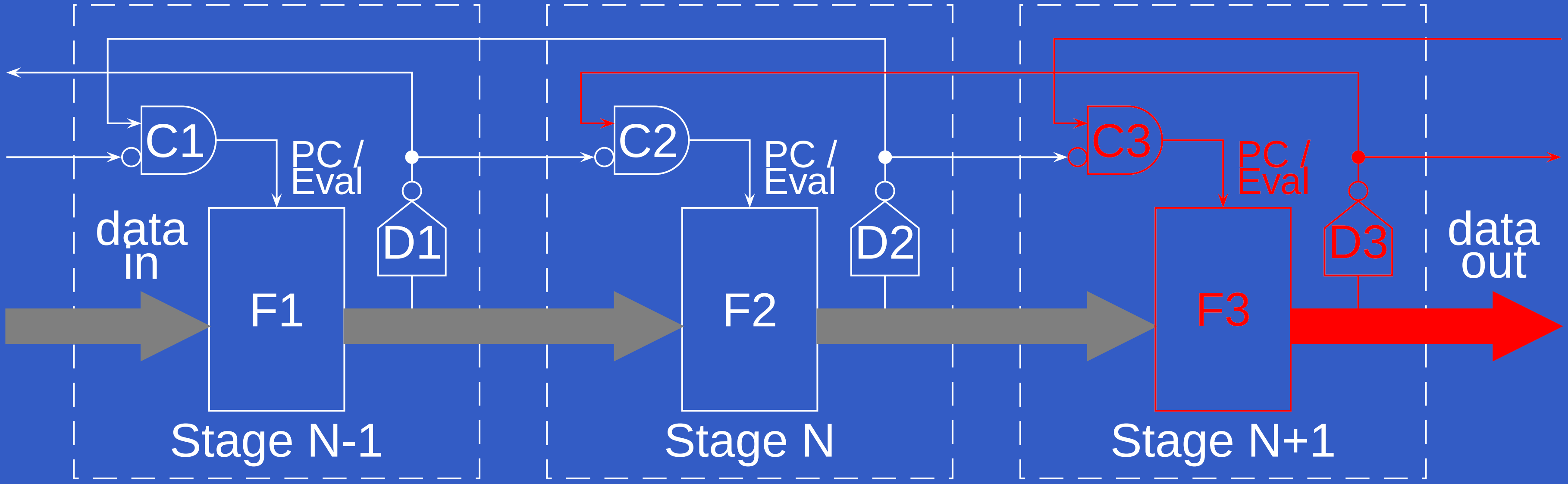
- stage N-1 has reset its data
- stage N+1 has received the data



PC0, Williams[1991]

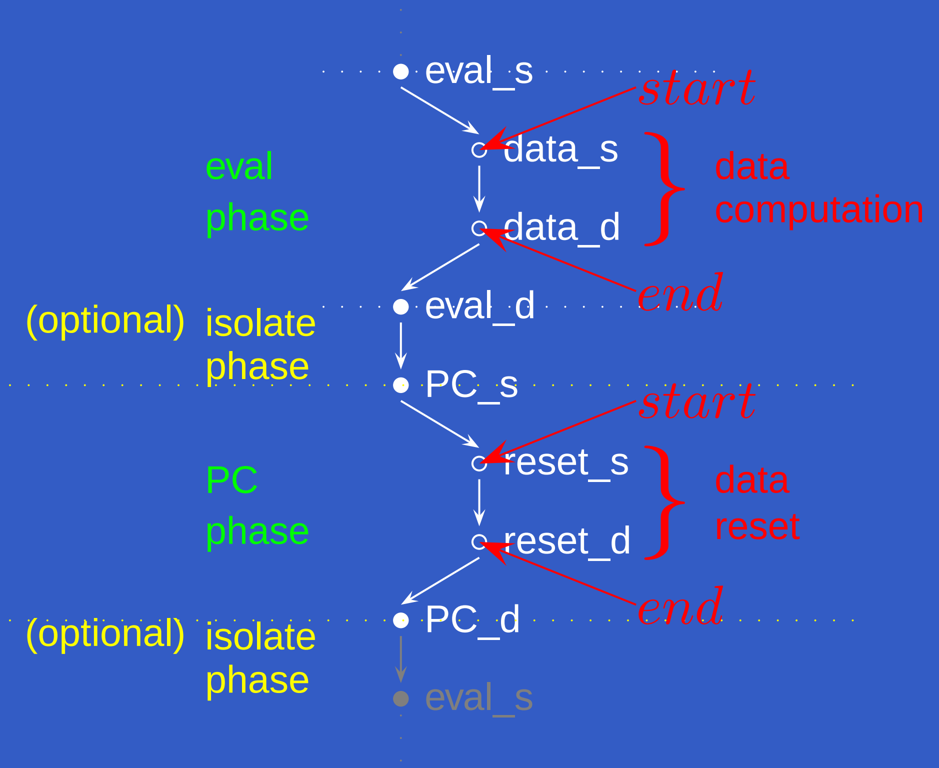
Asynchronous Pipeline: PC0

Stage N precharges

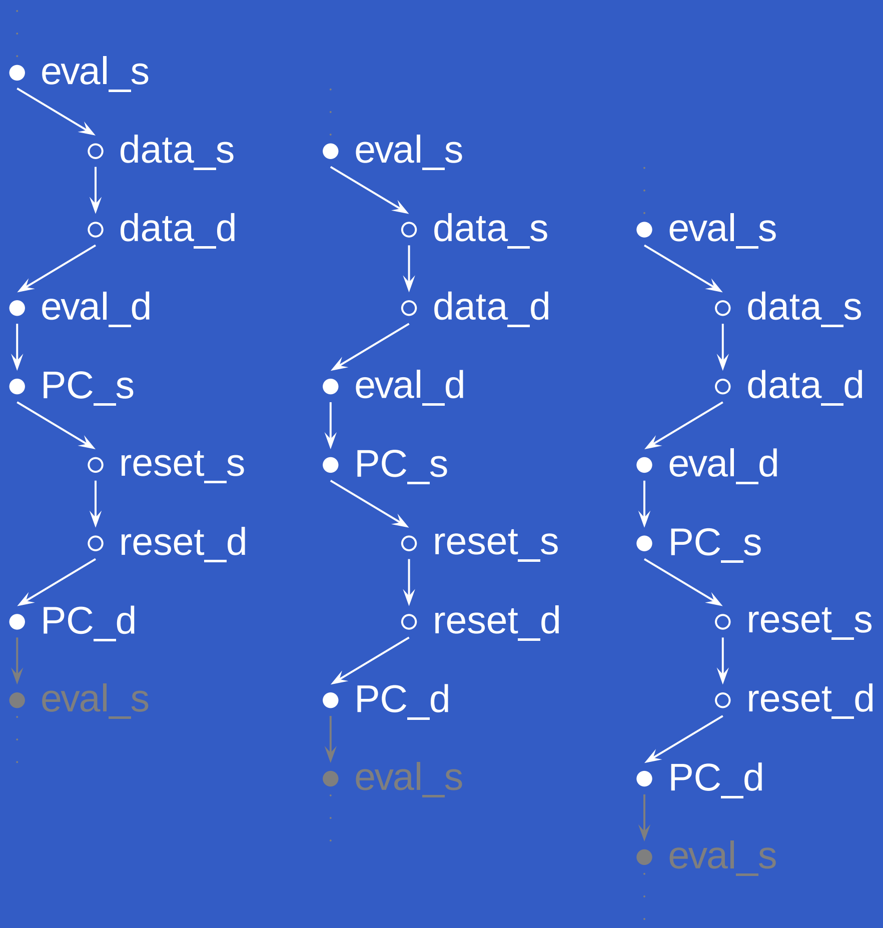


PC0, Williams[1991]

Protocol Graph: a Single Dynamic Pipeline Stage

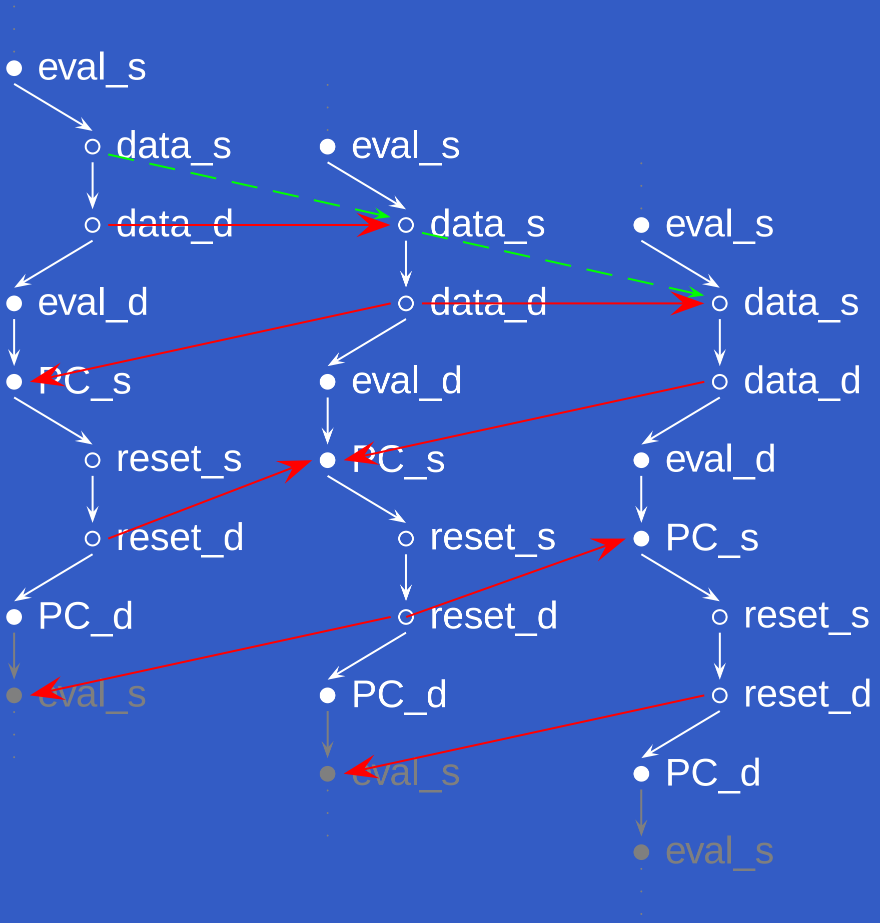


Protocol Graph: PC0



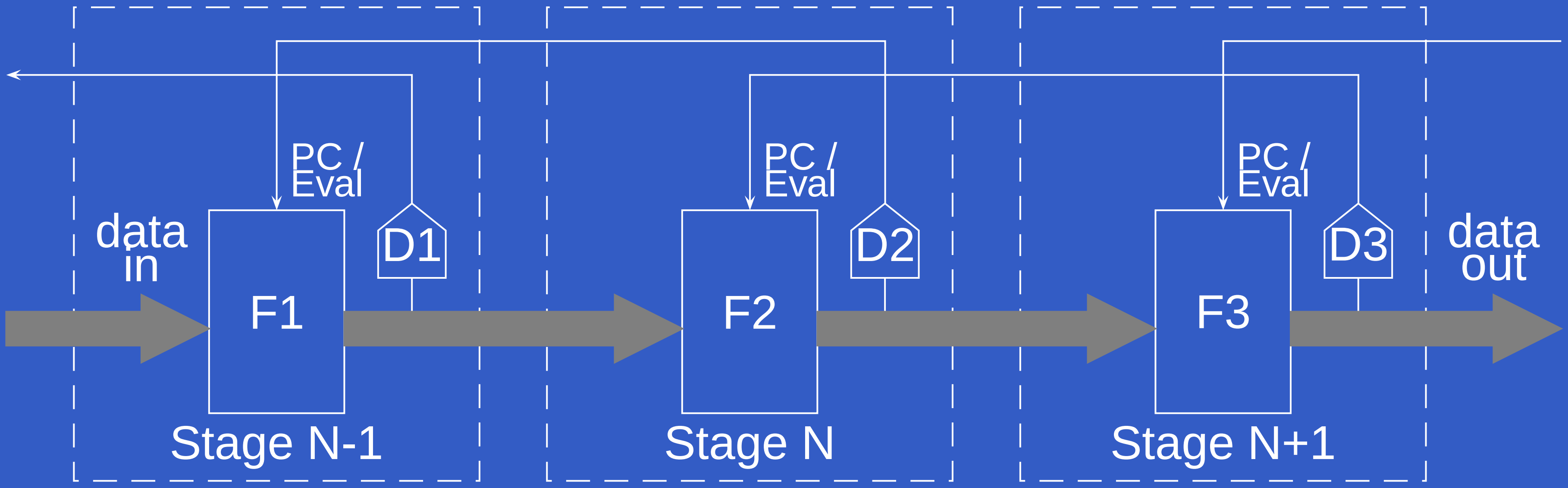
Stage N-1 Stage N Stage N+1

Protocol Graph: PC0



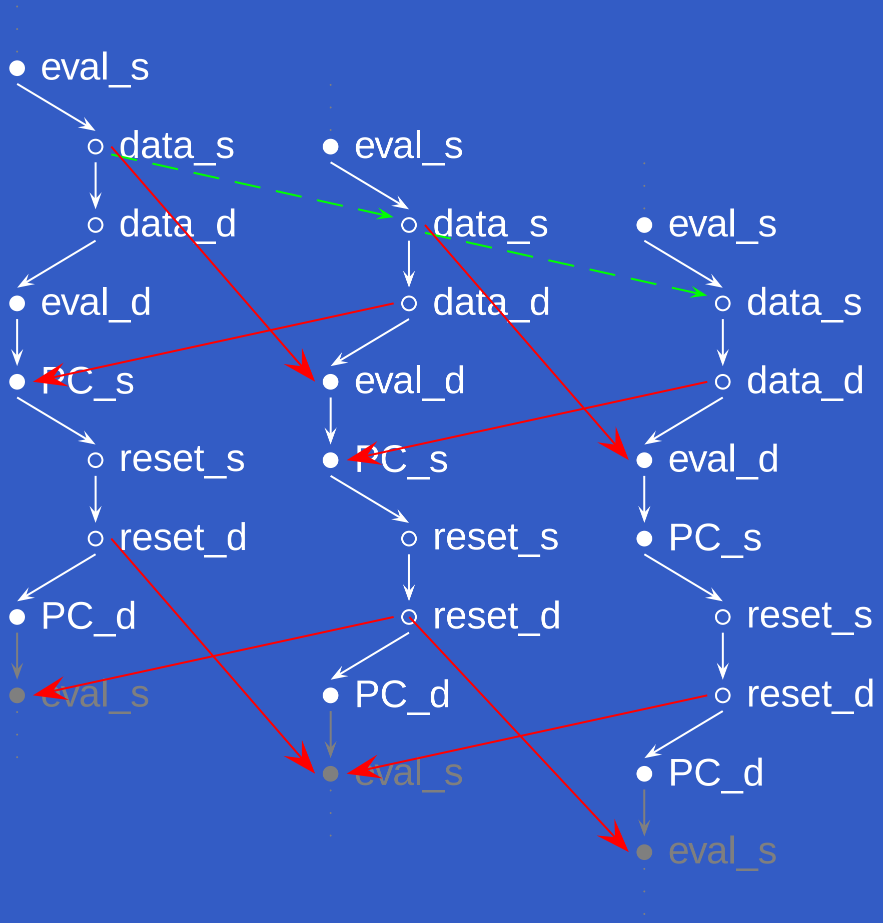
Stage N-1 Stage N Stage N+1

Asynchronous Pipeline: PS0



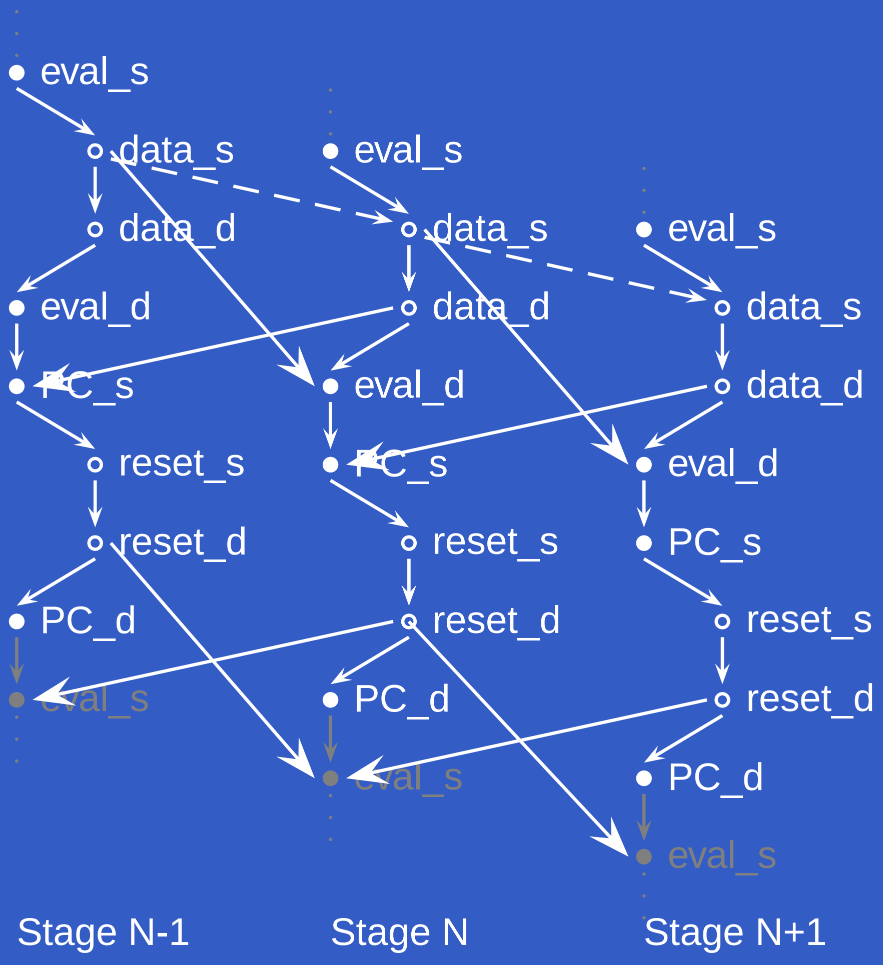
PS0, Williams[1991]

Protocol Graph: PS0



Stage N-1 Stage N Stage N+1

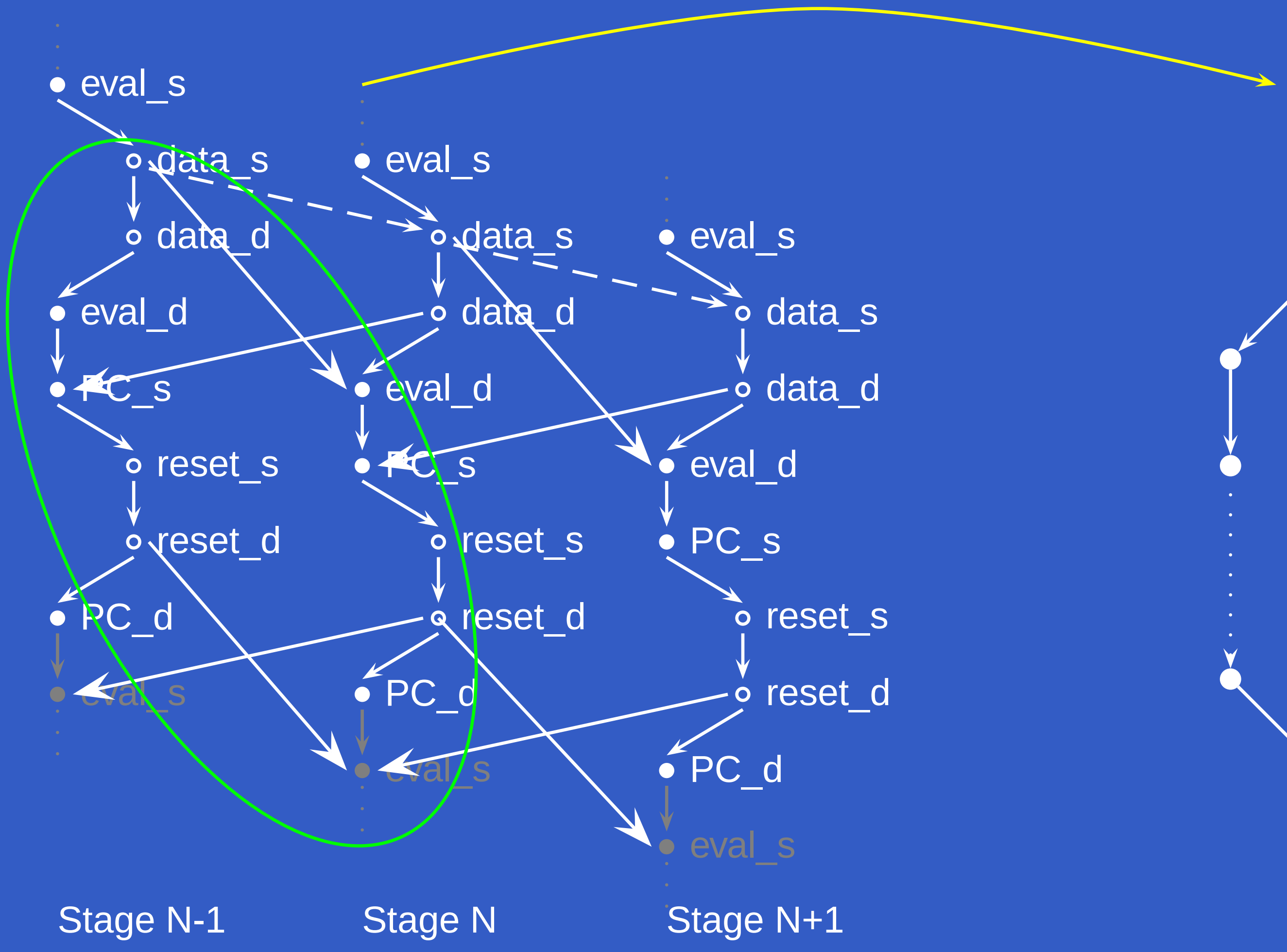
Graph Transformation: Basic Idea



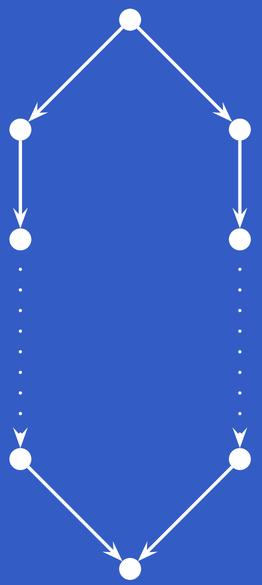
Protocol Graph

- interstage arcs control concurrency of protocol
- change concurrency by moving arcs

Graph Transformation: Basic Idea



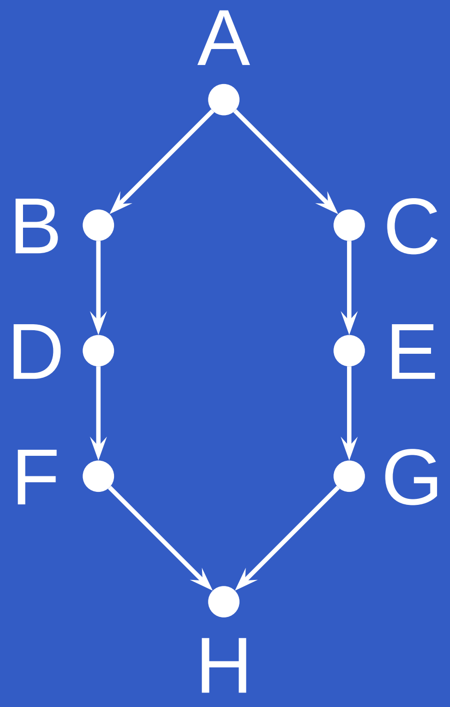
Protocol Graph



Extracted Sub-Graph

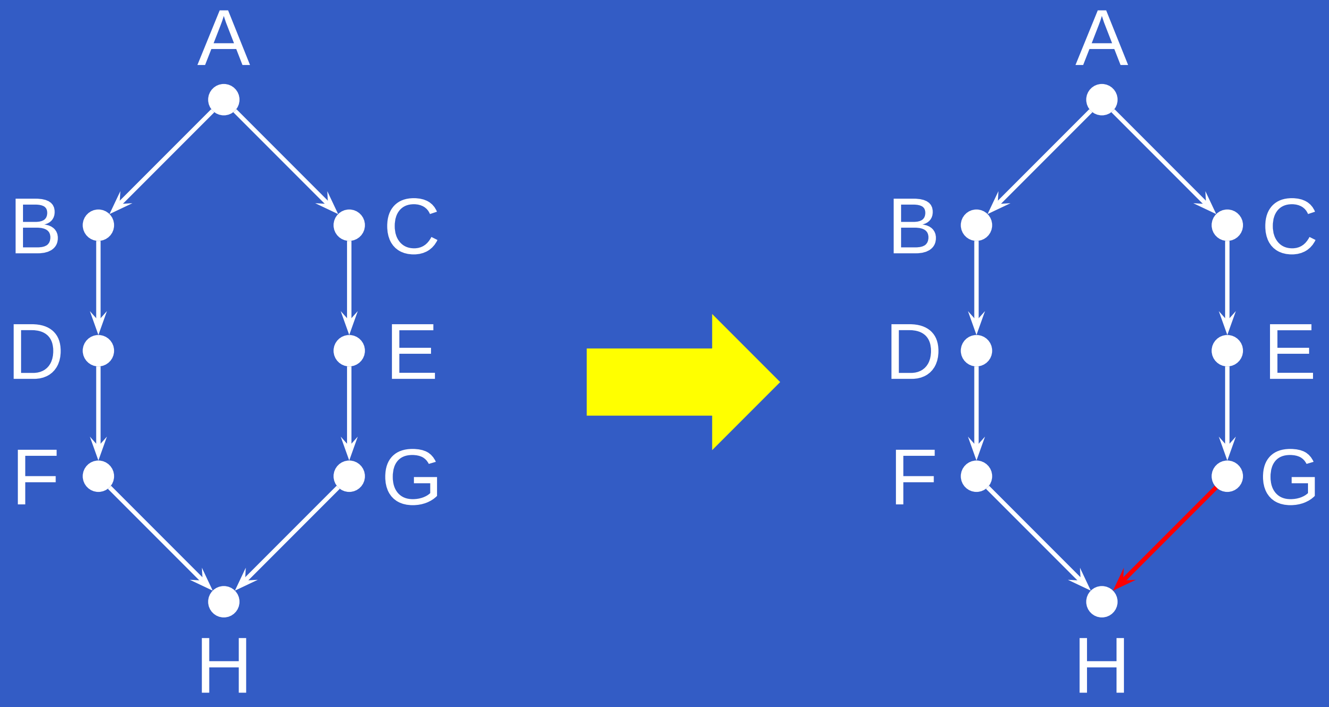
Transformation: Rule M1

Concurrency reduction move: move arrow target up



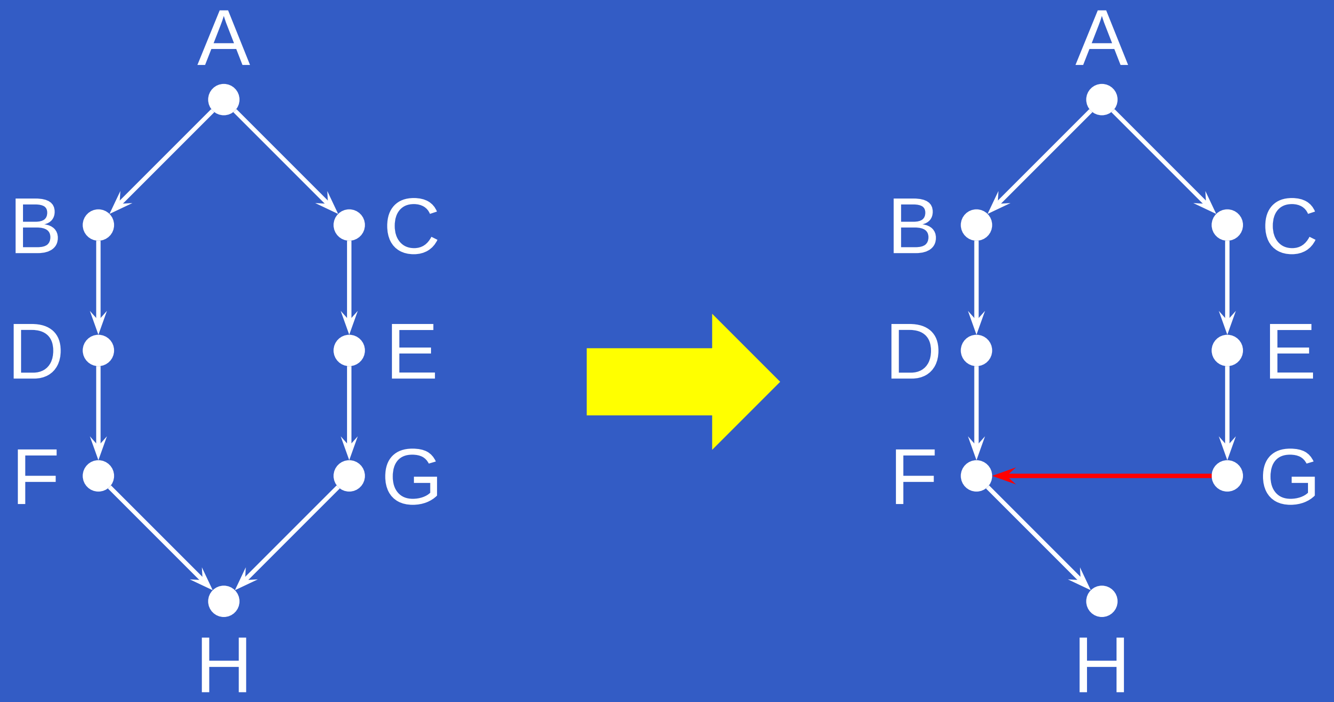
Transformation: Rule M1

Concurrency reduction move: move arrow target up


$$\begin{tikzcd}[column sep=small] & A \arrow[dl] \arrow[dr] & & & & A \arrow[dl] \arrow[dr] & \\ B \arrow[d] & & C \arrow[d] & & B \arrow[d] & & C \arrow[d] \\ D \arrow[d] & & E \arrow[d] & \Rightarrow & D \arrow[d] & & E \arrow[d] \\ F \arrow[dr] & & G \arrow[dl] & & F \arrow[dr] & & G \arrow[dl] \\ & H & & & & H & \end{tikzcd}$$

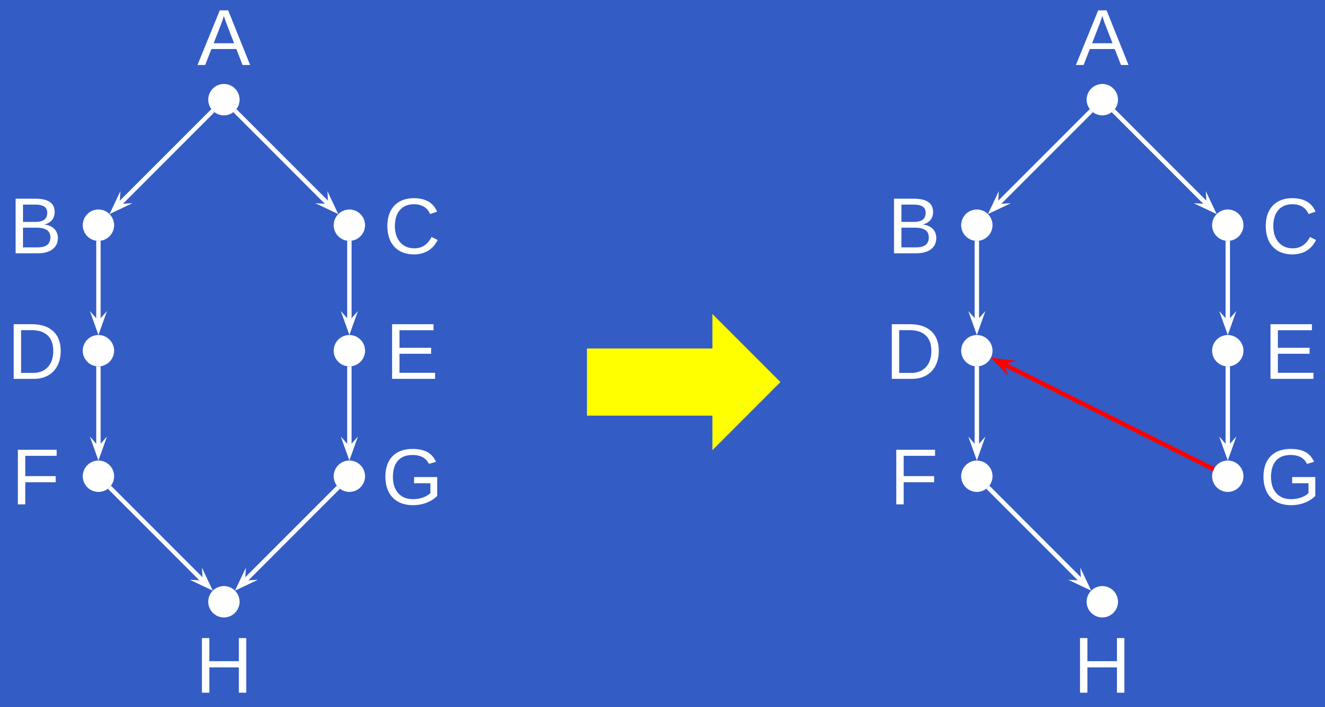
Transformation: Rule M1

Concurrency reduction move: move arrow target up



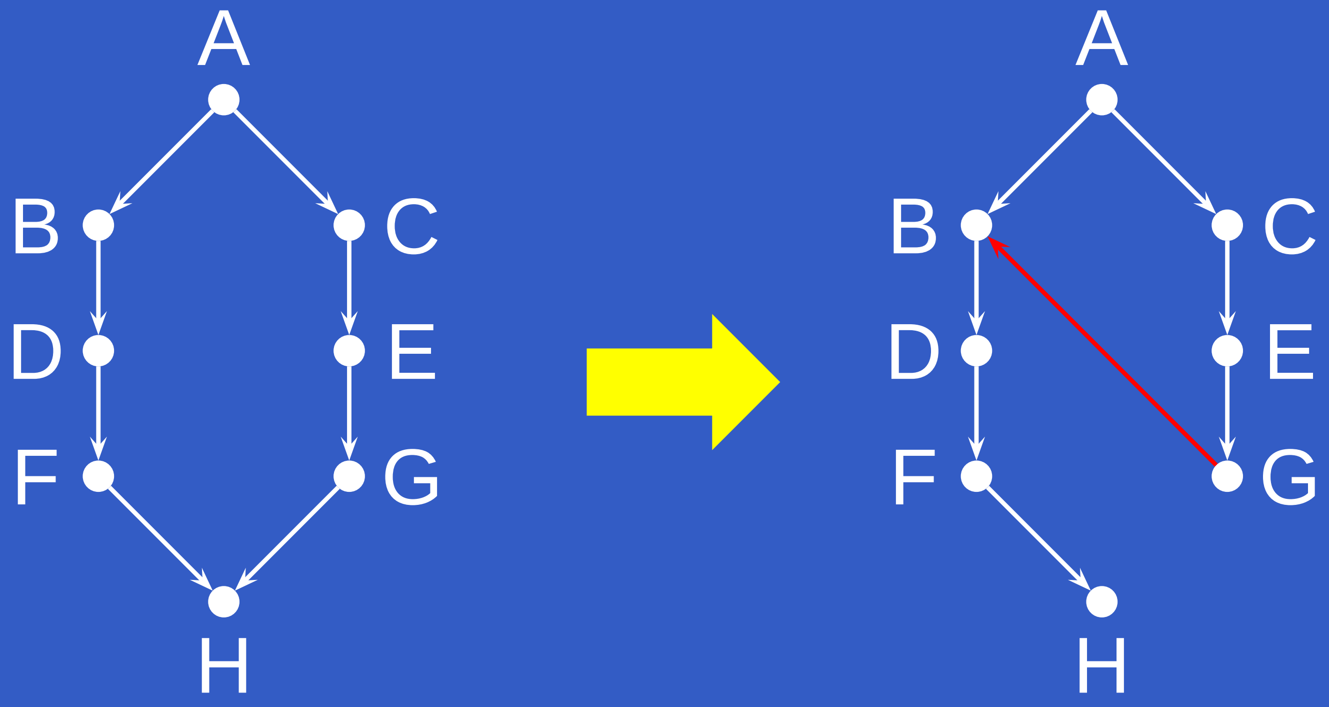
Transformation: Rule M1

Concurrency reduction move: move arrow target up



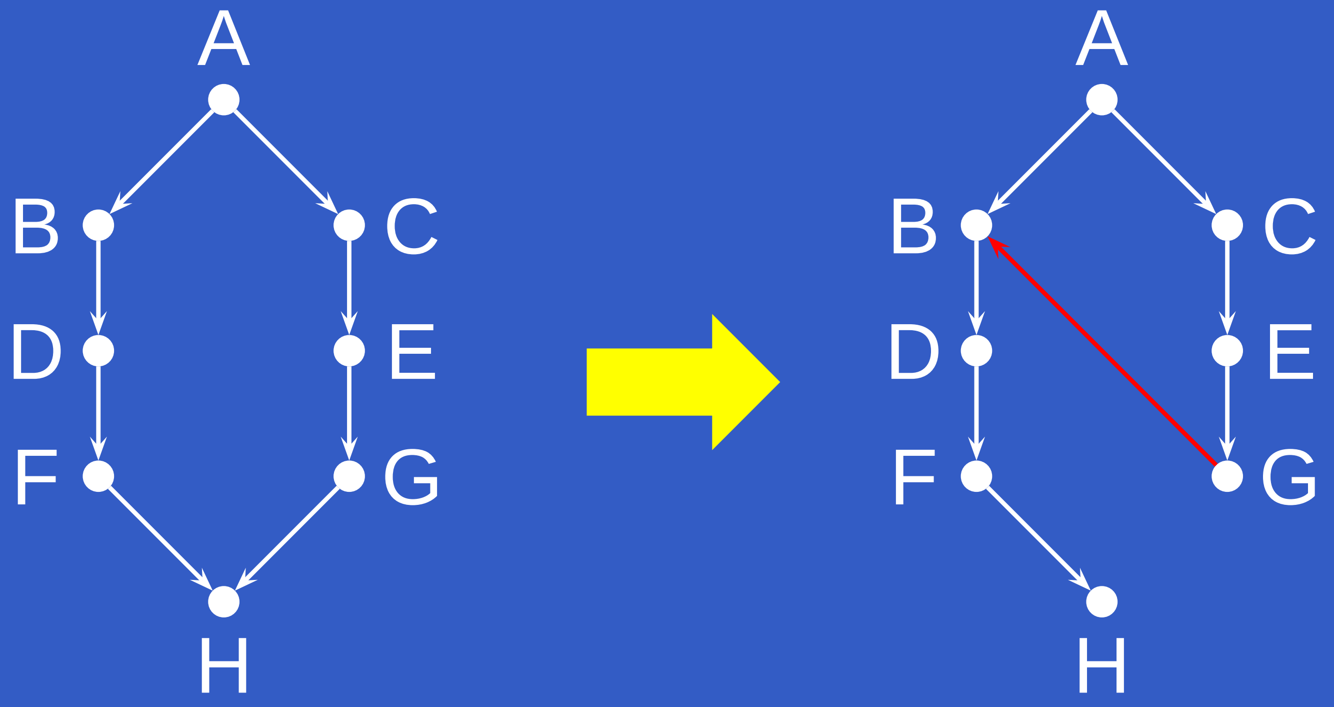
Transformation: Rule M1

Concurrency reduction move: move arrow target up



Transformation: Rule M1

Concurrency reduction move: move arrow target up

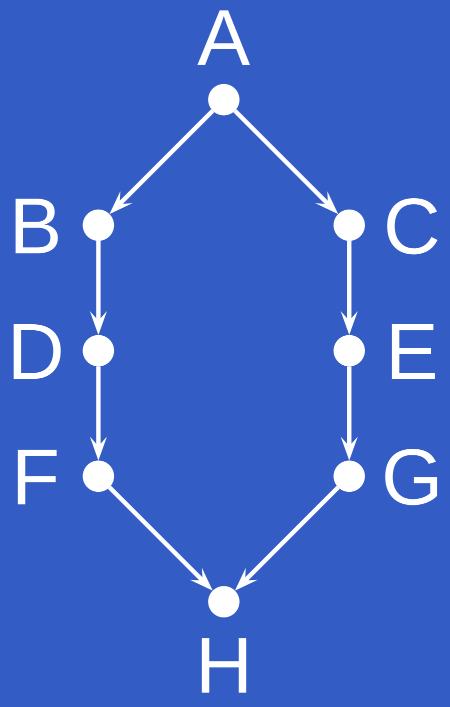


$A \prec B \prec D \prec F \prec H$
and
$A \prec C \prec E \prec G \prec H$

$A \prec B \prec D \prec F \prec H$
and
$A \prec C \prec E \prec B \prec D \prec F \prec H$

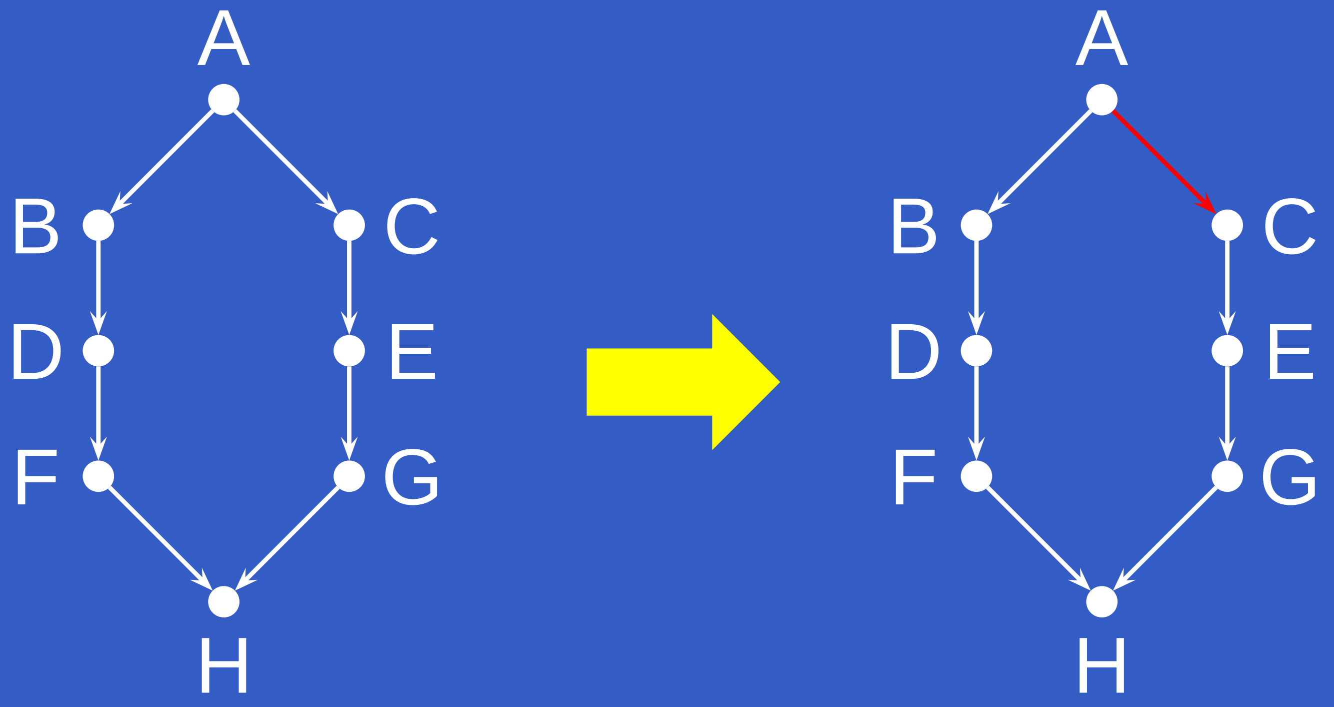
Transformations: Rule M2

Concurrency reduction move: move arrow origin down



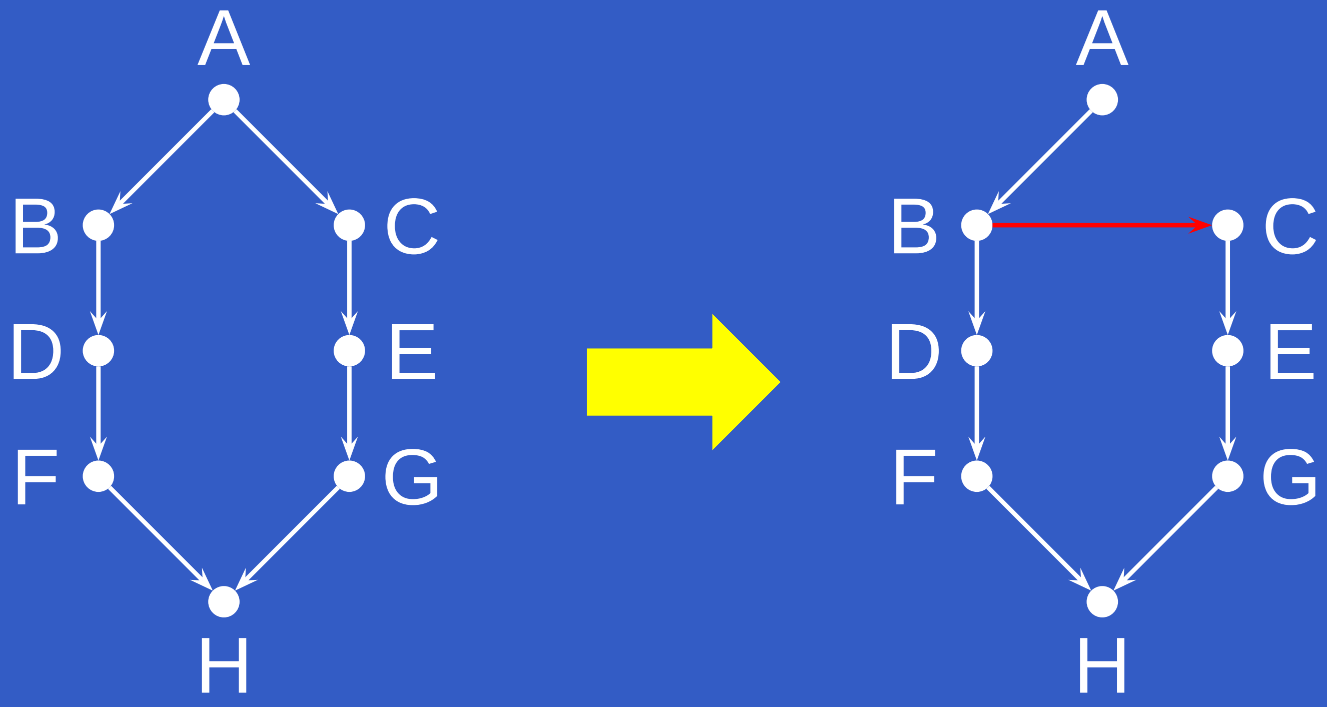
Transformations: Rule M2

Concurrency reduction move: move arrow origin down



Transformations: Rule M2

Concurrency reduction move: move arrow origin down



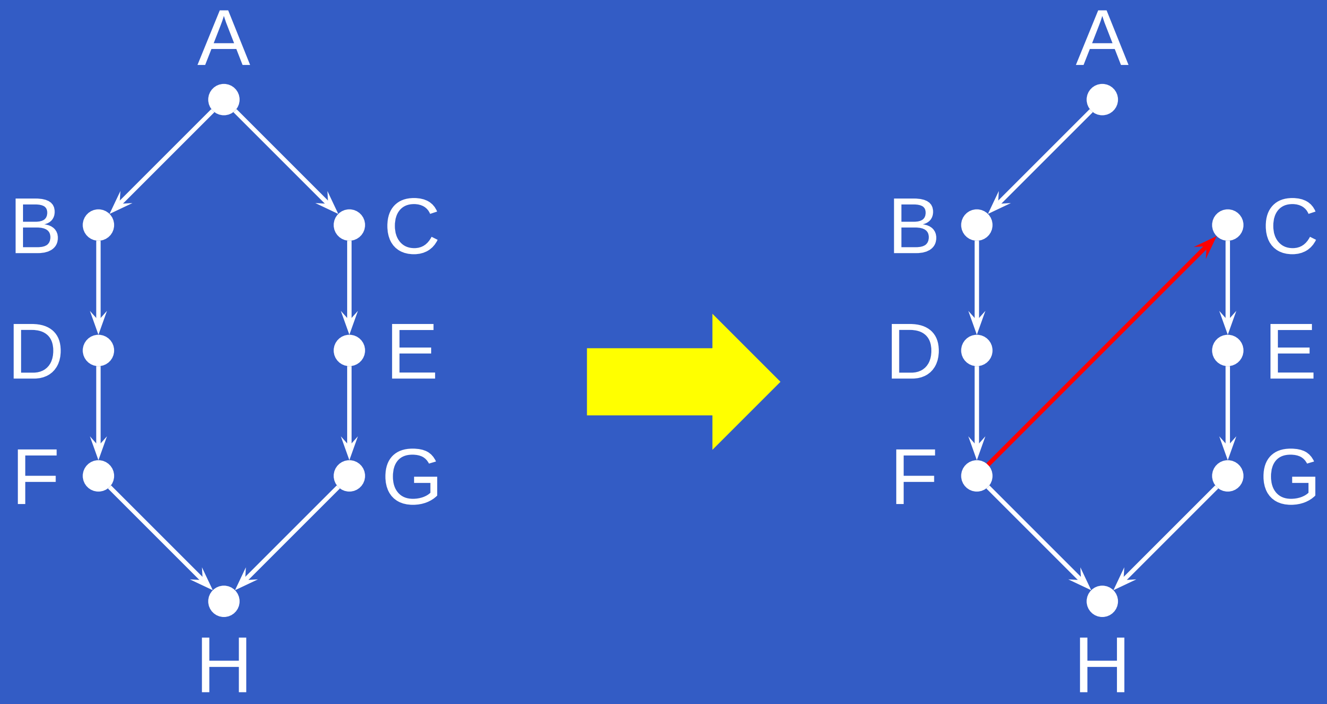
Transformations: Rule M2

Concurrency reduction move: move arrow origin down

$$\begin{tikzcd}
& A \arrow[dl] \arrow[dr] & & & & A \arrow[dl] & \\
B \arrow[d] & & C \arrow[d] & & B \arrow[d] & & C \arrow[d] \\
D \arrow[d] & & E \arrow[d] & \Longrightarrow & D \arrow[d] \arrow[urr] & & E \arrow[d] \\
F \arrow[dr] & & G \arrow[dl] & & F \arrow[dr] & & G \arrow[dl] \\
& H & & & & H &
\end{tikzcd}$$

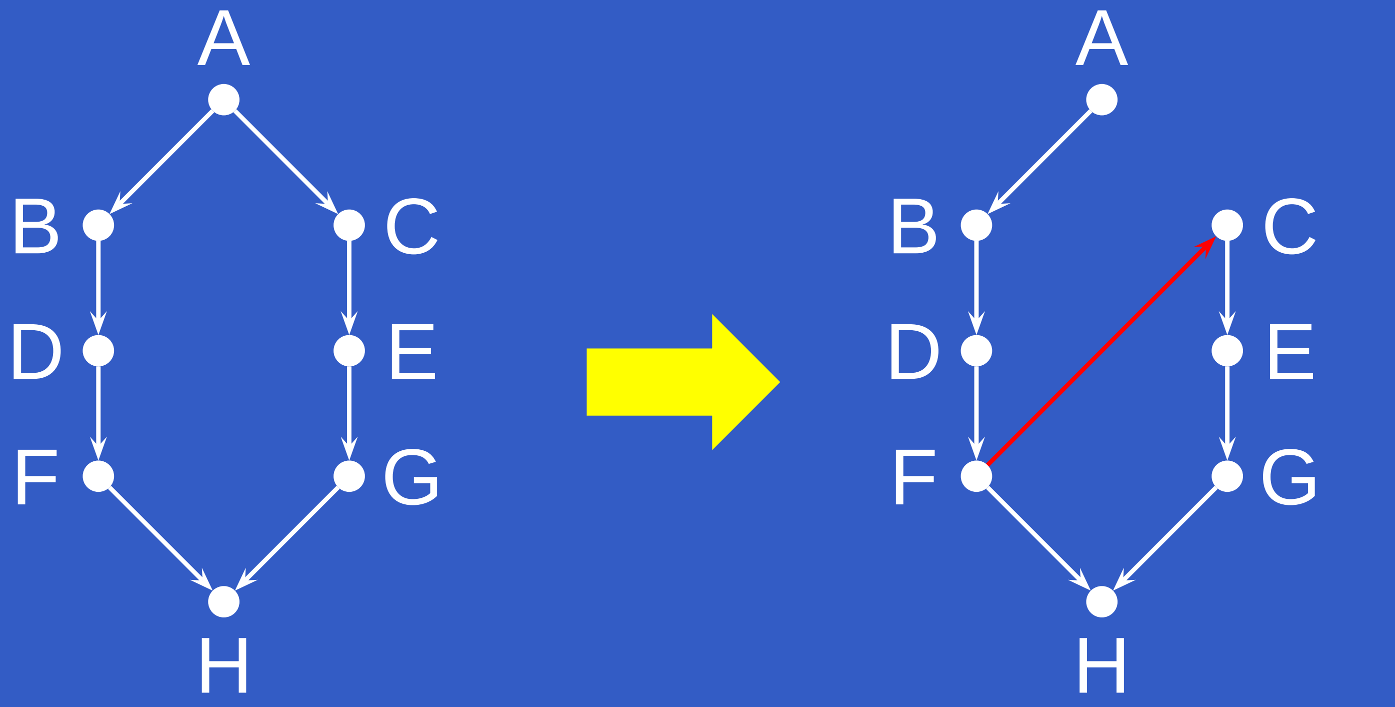
Transformations: Rule M2

Concurrency reduction move: move arrow origin down



Transformations: Rule M2

Concurrency reduction move: move arrow origin down



$A \prec B \prec D \prec F \prec H$
and
$A \prec C \prec E \prec G \prec H$

$A \prec B \prec D \prec F \prec H$
and
$F \prec C \prec E \prec G \prec H$

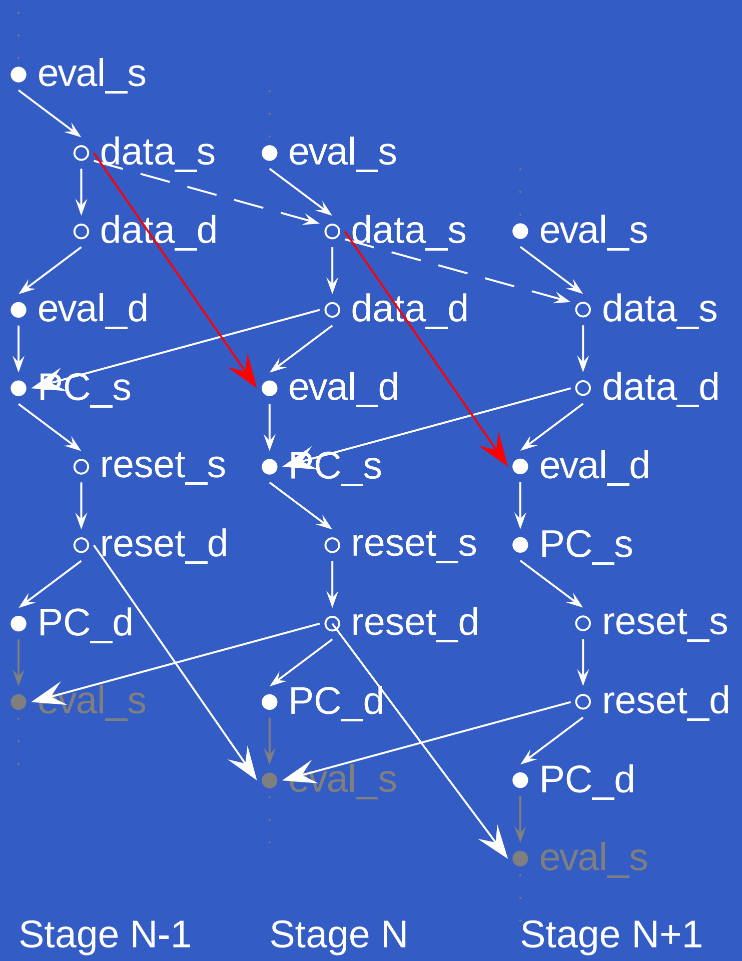
Transformations: Rule M3

Redundancy removal

$$\begin{tikzcd} A \arrow[d] \arrow[dd, bend left] & & A \arrow[d] \\ B \arrow[d] & \Rightarrow & B \arrow[d] \\ C & & C \end{tikzcd}$$

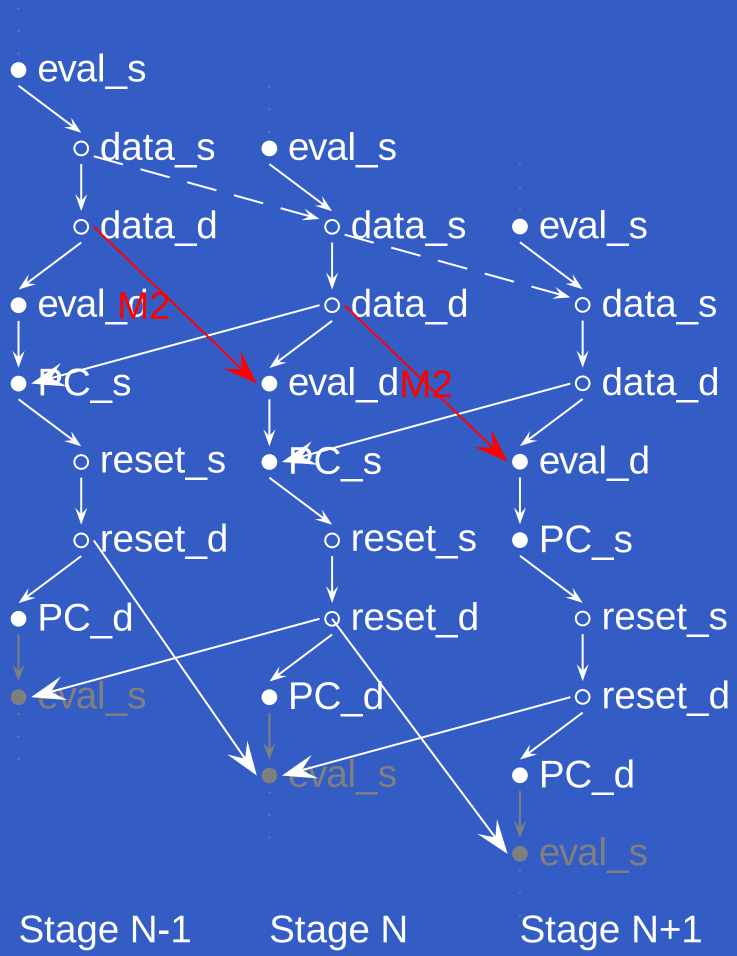
From PS0 to PC0

Applying transformation rules
to protocol graph:



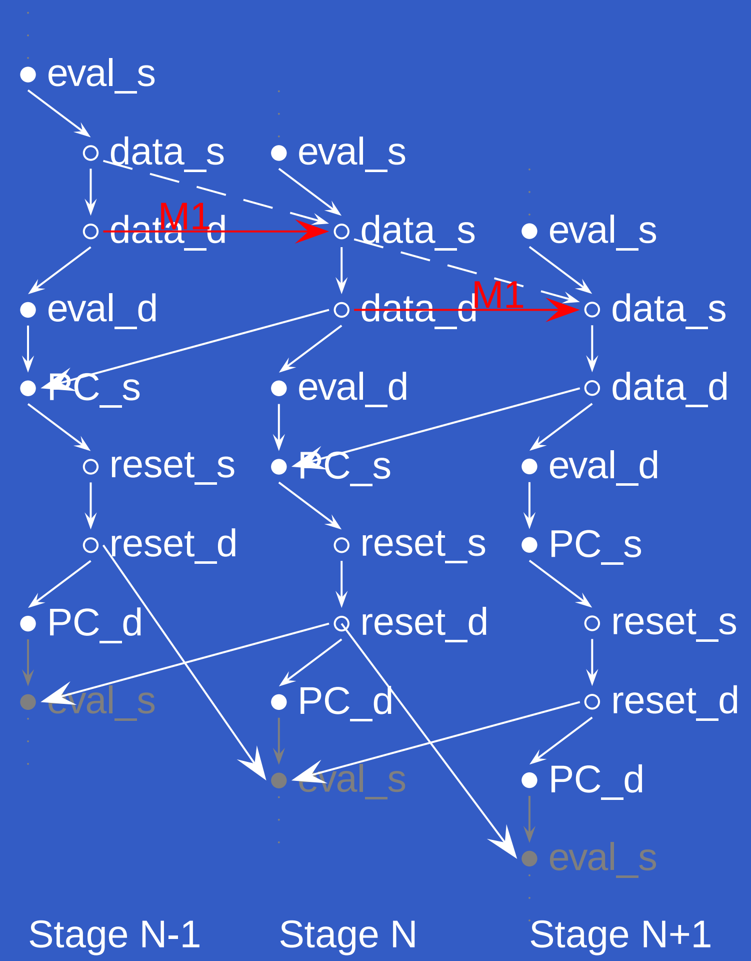
From PS0 to PC0

Applying transformation rules
to protocol graph:



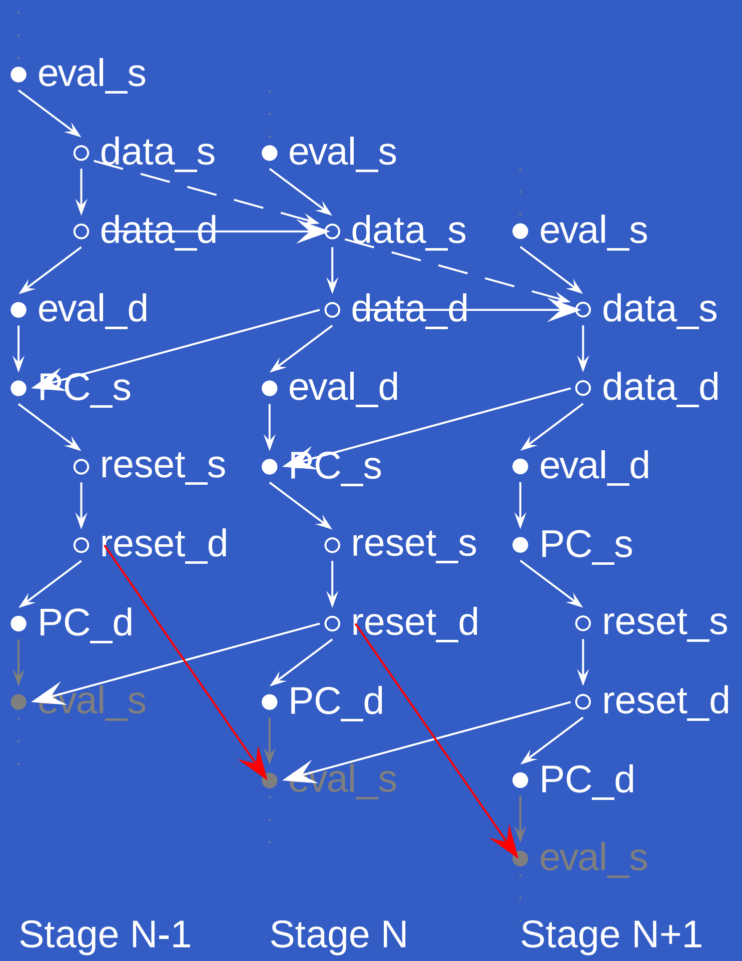
From PS0 to PC0

Applying transformation rules
to protocol graph:



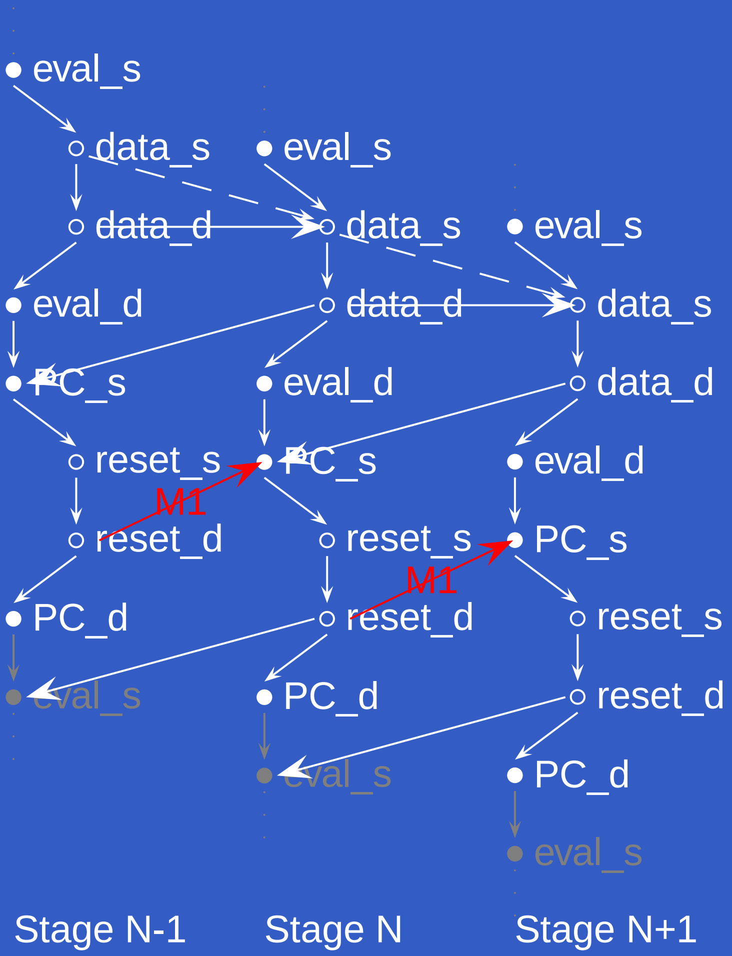
From PS0 to PC0

Applying transformation rules
to protocol graph:



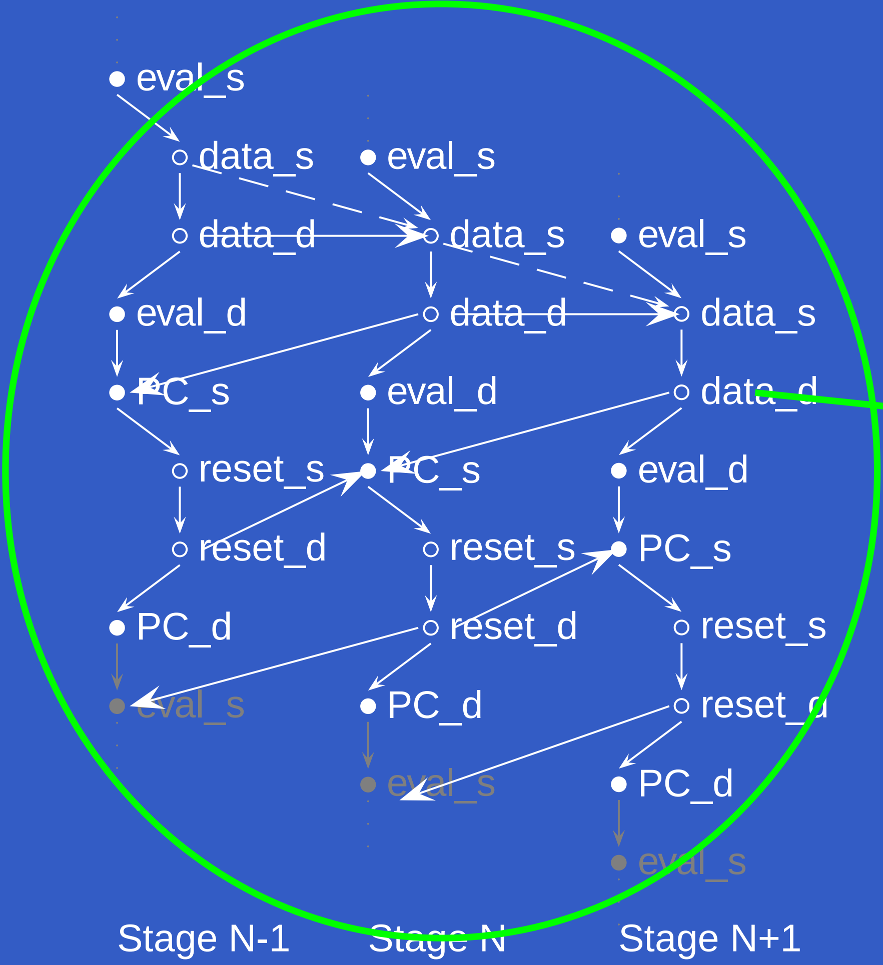
From PS0 to PC0

Applying transformation rules
to protocol graph:

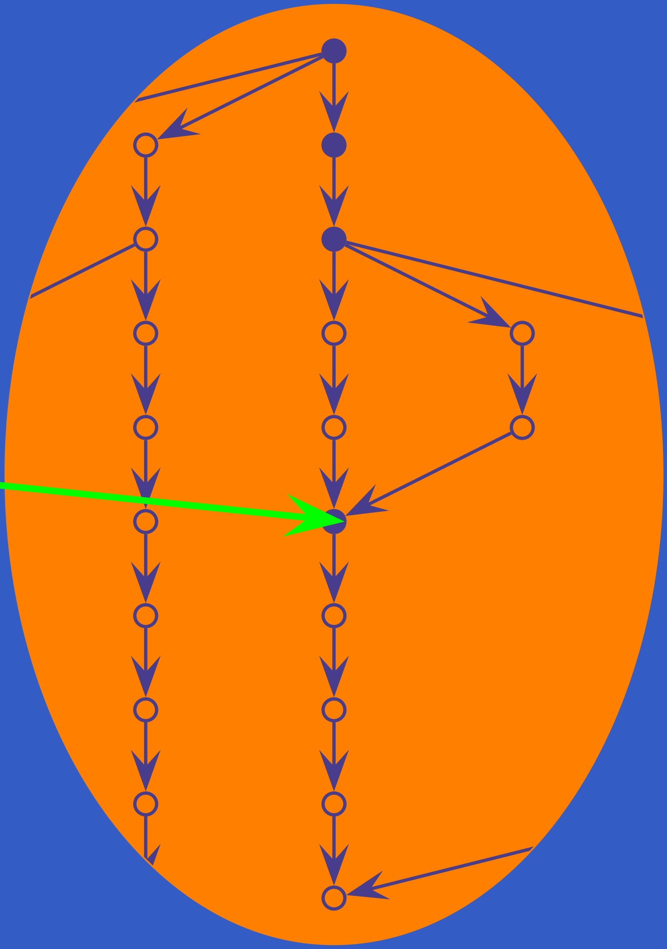


Deriving a Lattice for the Design Space

Applying transformation rules
to protocol graph:



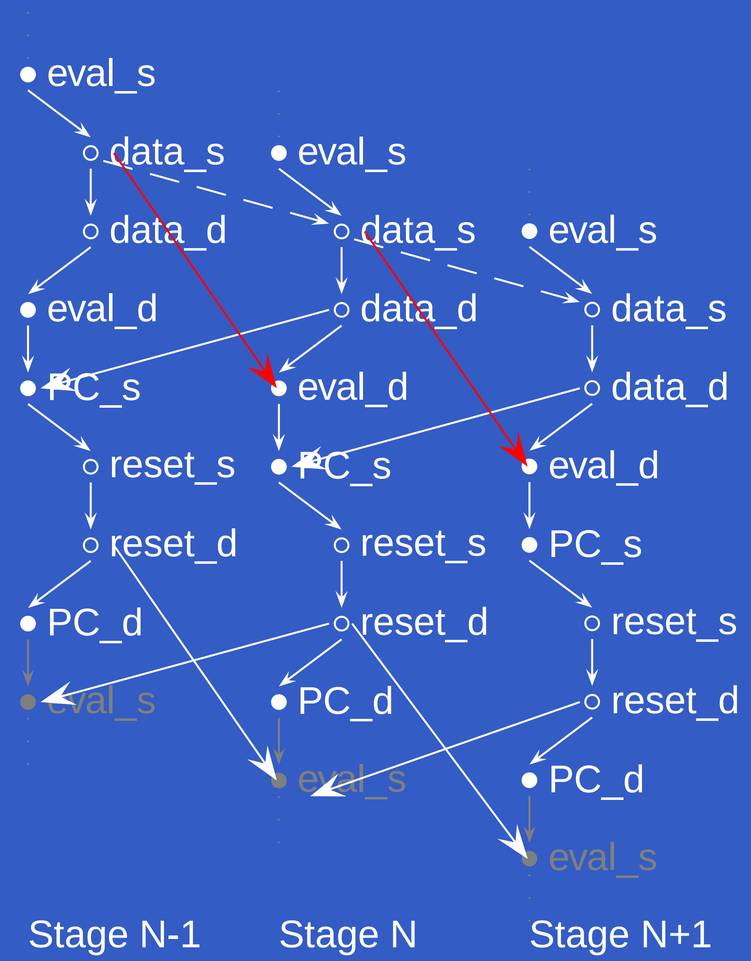
Equivalent to walking down semi-lattice



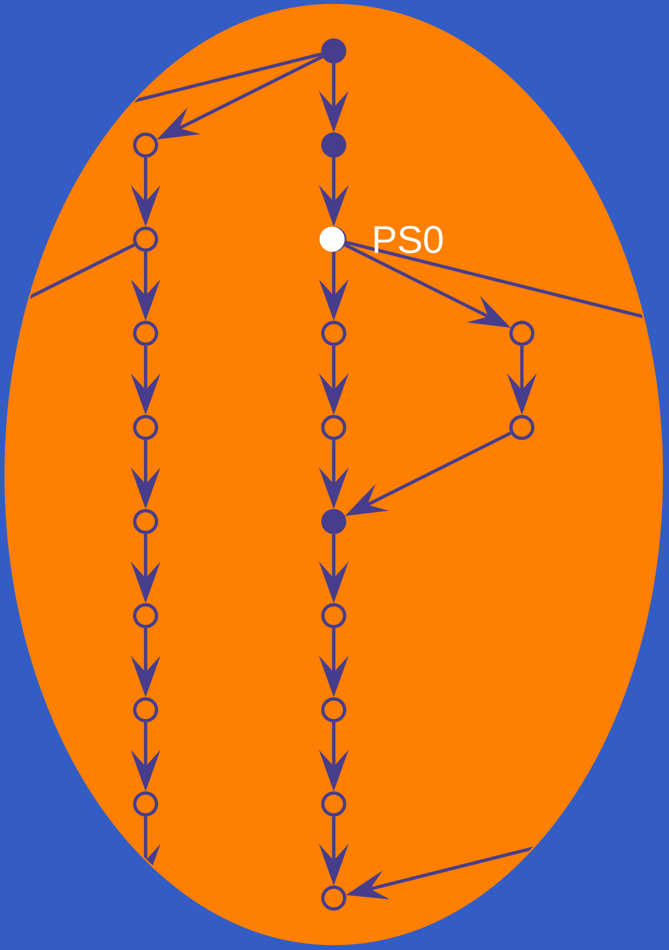
Semi-lattice = design space of pipeline protocols

Deriving a Lattice for the Design Space

Applying transformation rules
to protocol graph:



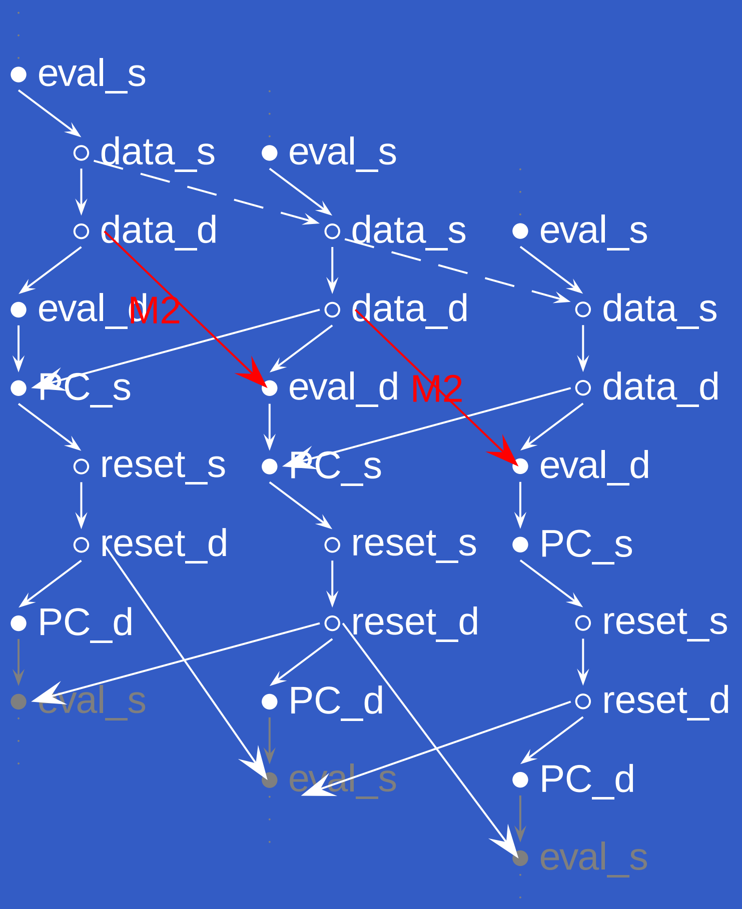
Equivalent to walking down semi-lattice



Semi-lattice = design space of pipeline protocols

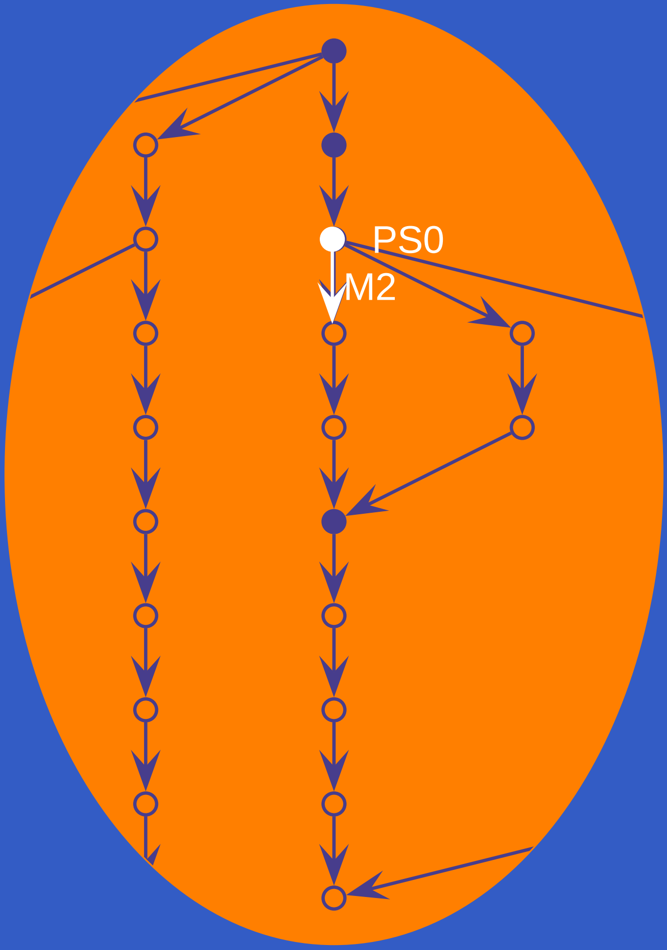
Deriving a Lattice for the Design Space

Applying transformation rules
to protocol graph:



Stage N-1 Stage N Stage N+1

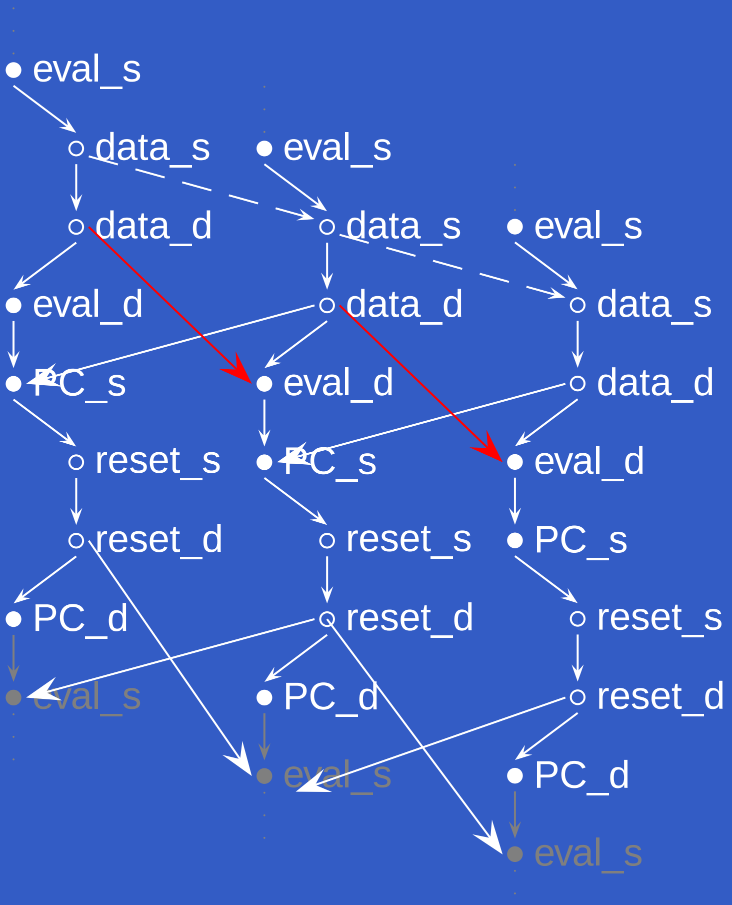
Equivalent to walking down semi-lattice



Semi-lattice = design space of pipeline protocols

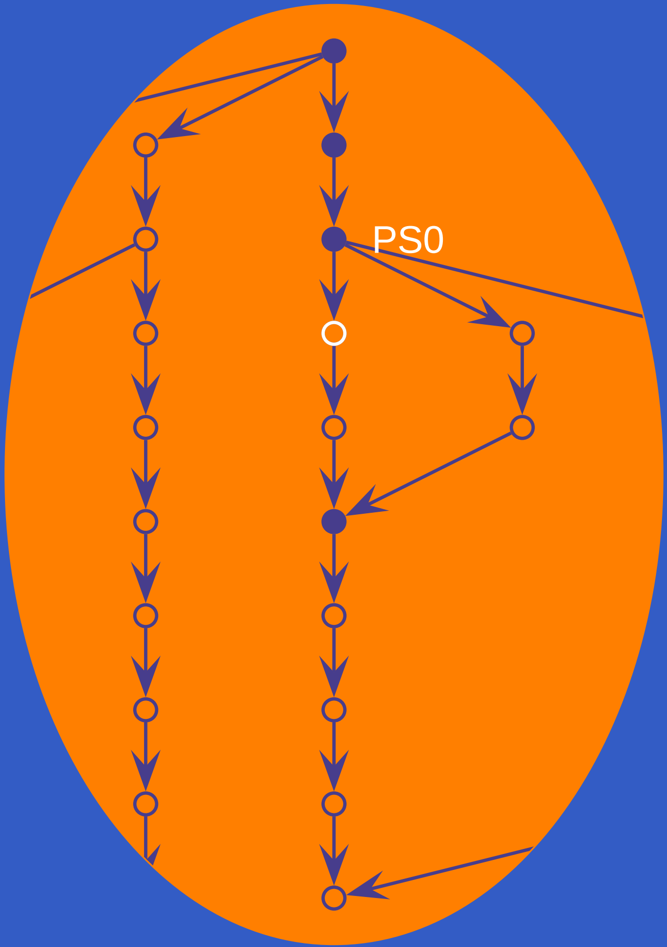
Deriving a Lattice for the Design Space

Applying transformation rules
to protocol graph:



Stage N-1 Stage N Stage N+1

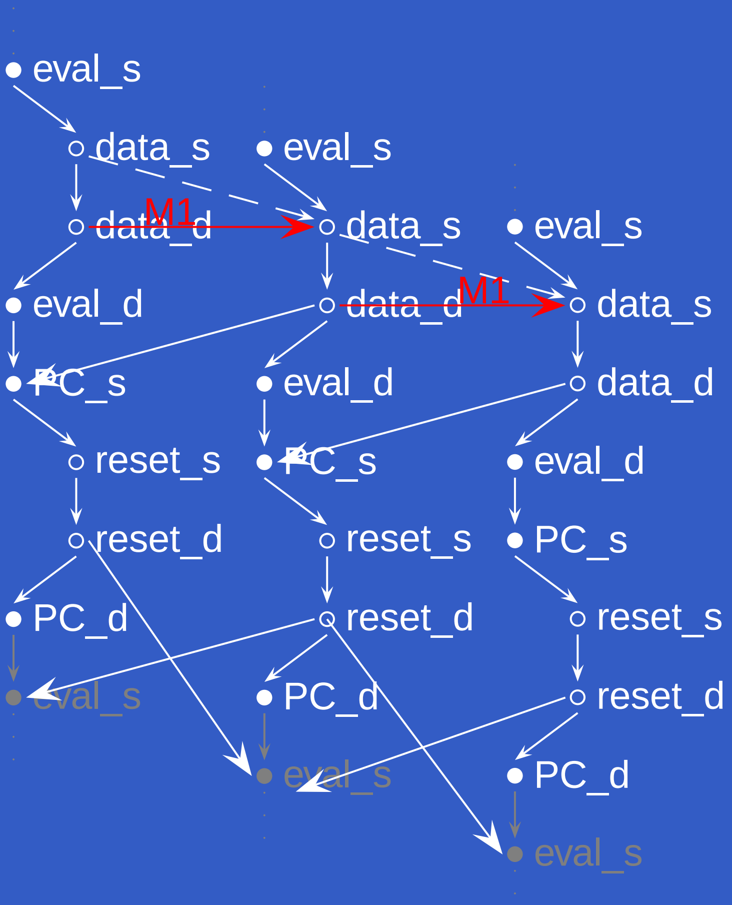
Equivalent to walking down semi-lattice



Semi-lattice = design space of pipeline protocols

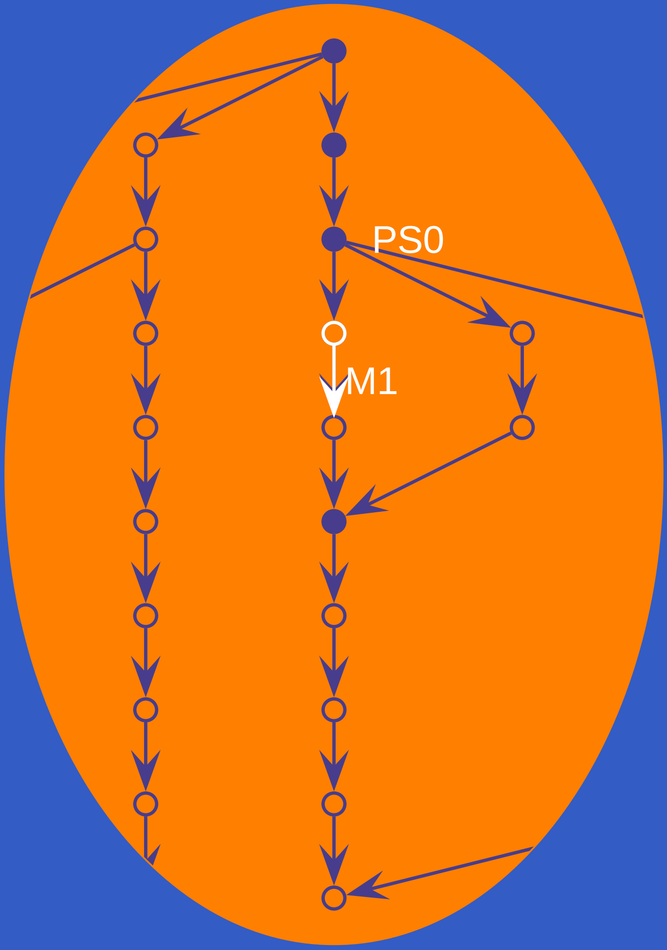
Deriving a Lattice for the Design Space

Applying transformation rules
to protocol graph:



Stage N-1 Stage N Stage N+1

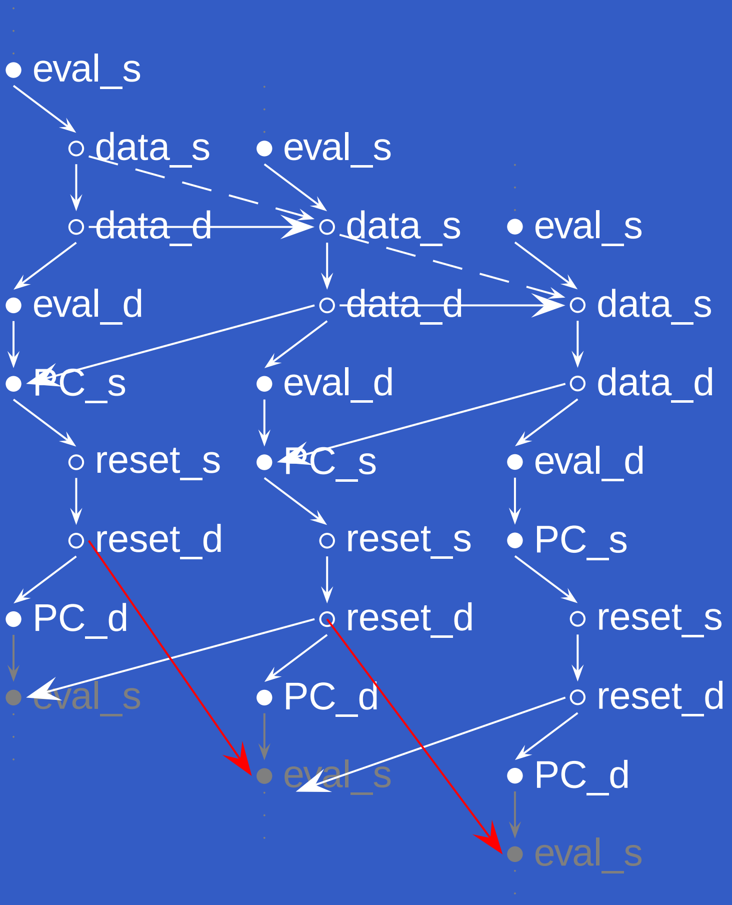
Equivalent to walking down semi-lattice



Semi-lattice = design space of pipeline protocols

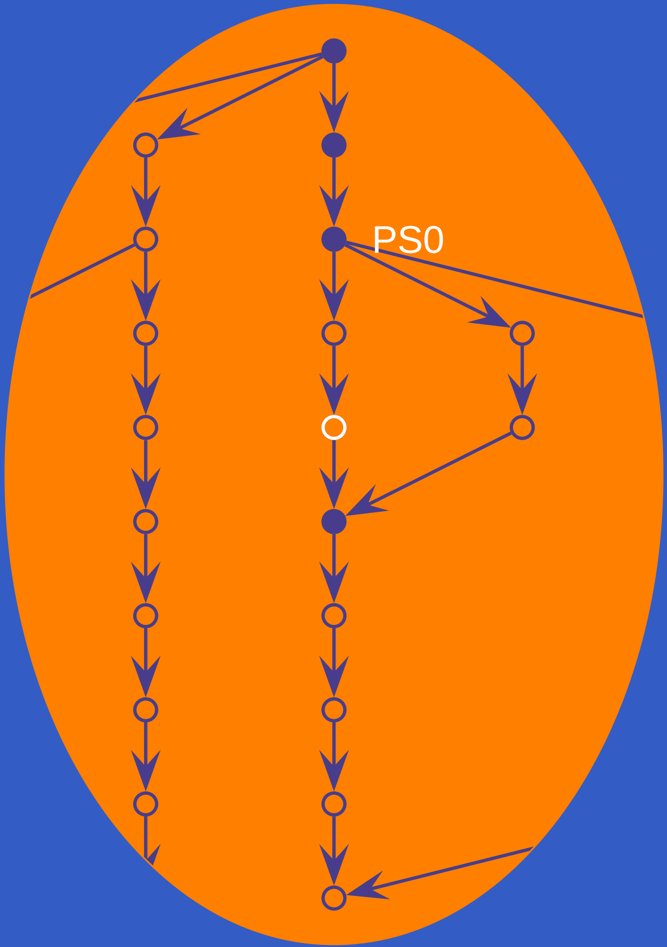
Deriving a Lattice for the Design Space

Applying transformation rules
to protocol graph:



Stage N-1 Stage N Stage N+1

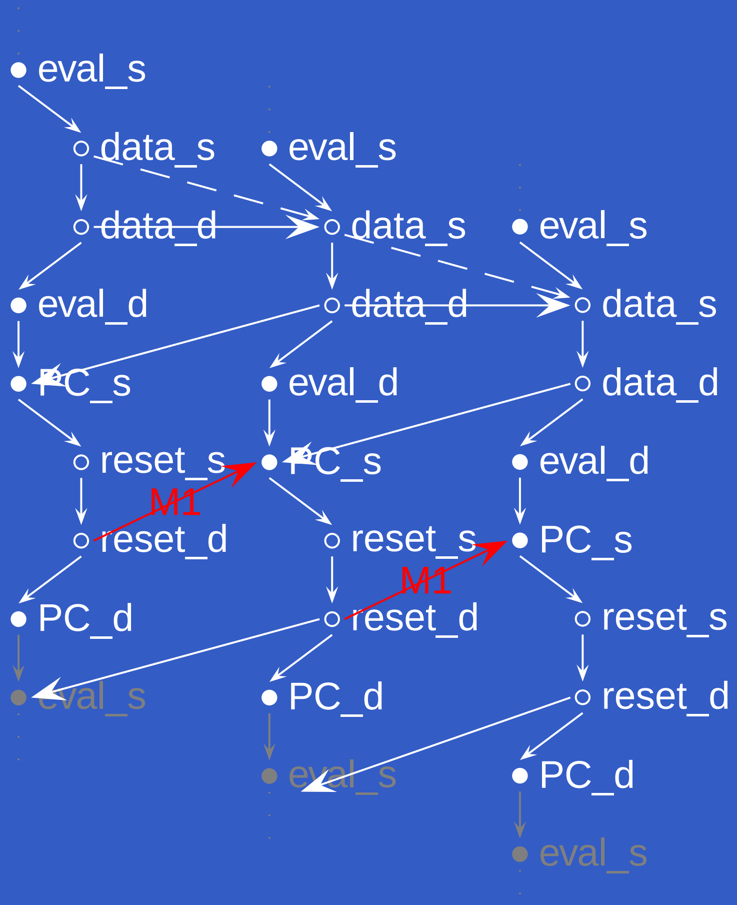
Equivalent to walking down semi-lattice



Semi-lattice = design space of pipeline protocols

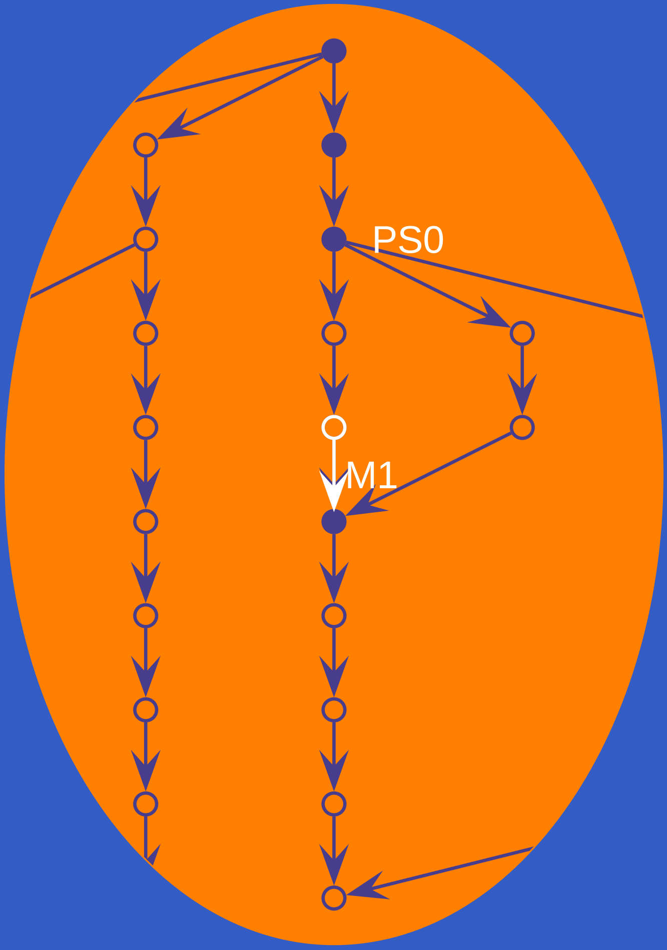
Deriving a Lattice for the Design Space

Applying transformation rules to protocol graph:



Stage N-1 Stage N Stage N+1

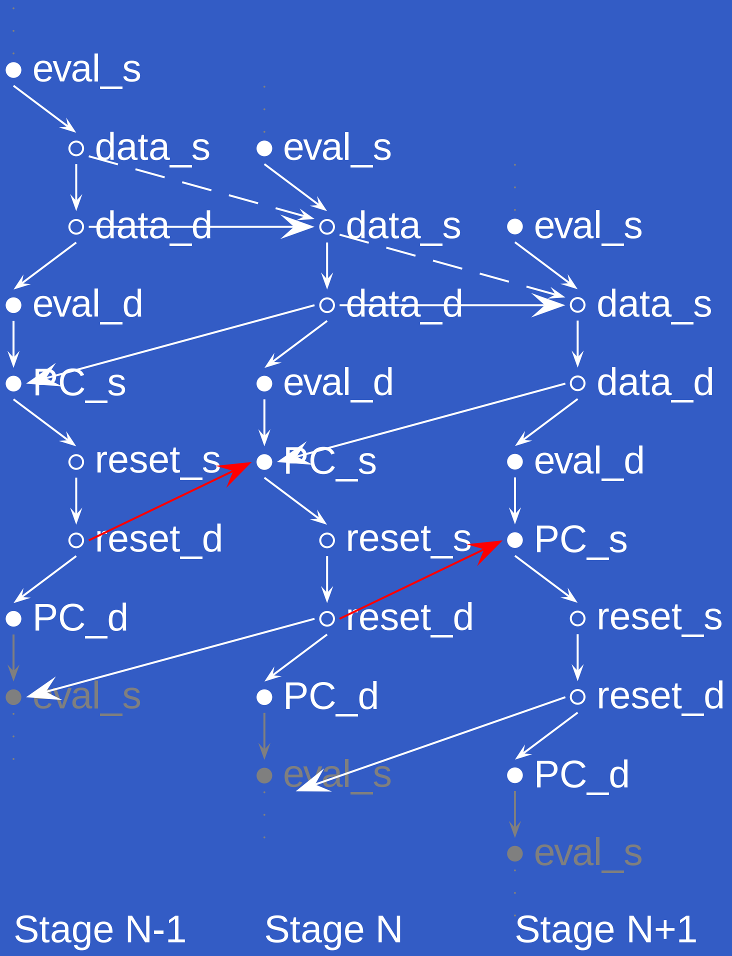
Equivalent to walking down semi-lattice



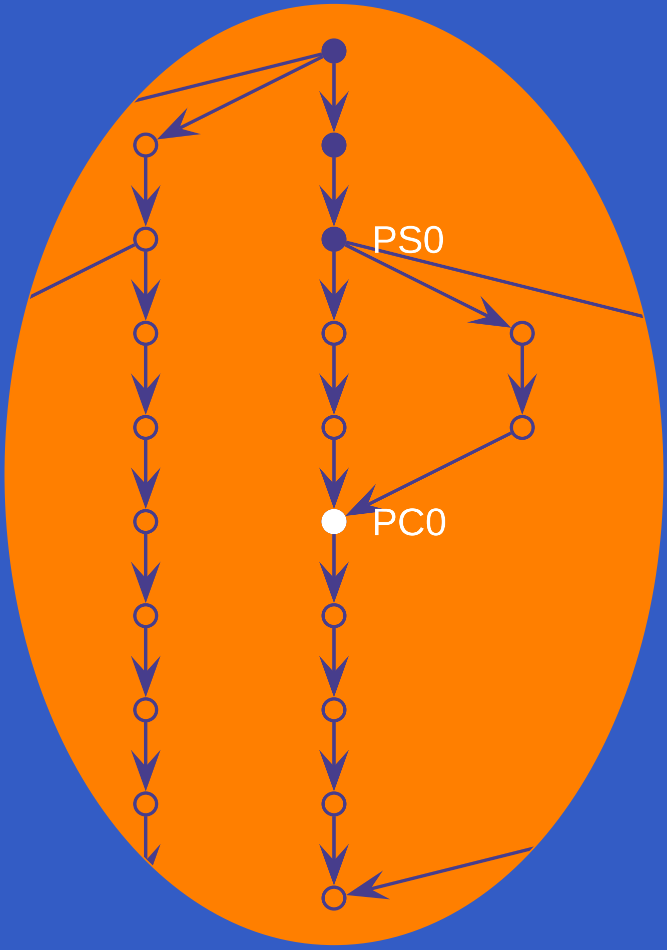
Semi-lattice = design space of pipeline protocols

Deriving a Lattice for the Design Space

Applying transformation rules to protocol graph:



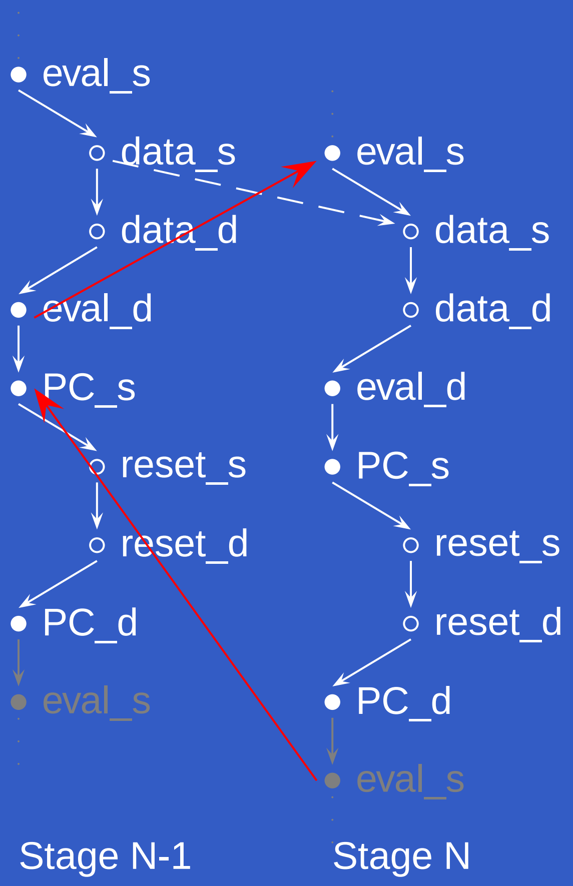
Equivalent to walking down semi-lattice



Semi-lattice = design space of pipeline protocols

How Far Can You Move?

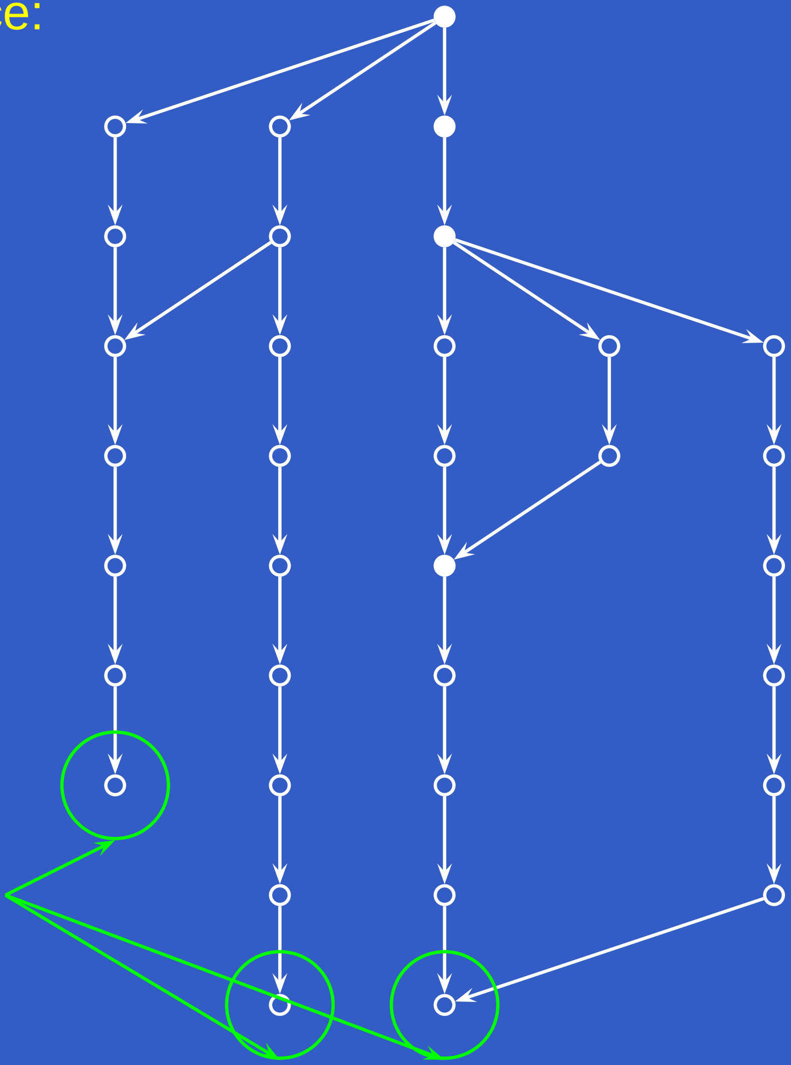
Protocol Graph:



- There exists a limit to the number of moves one can make
- Moves beyond that results in deadlocks
- Corresponds to the "bottoms" of the semi-lattice

How Far Can You Move?

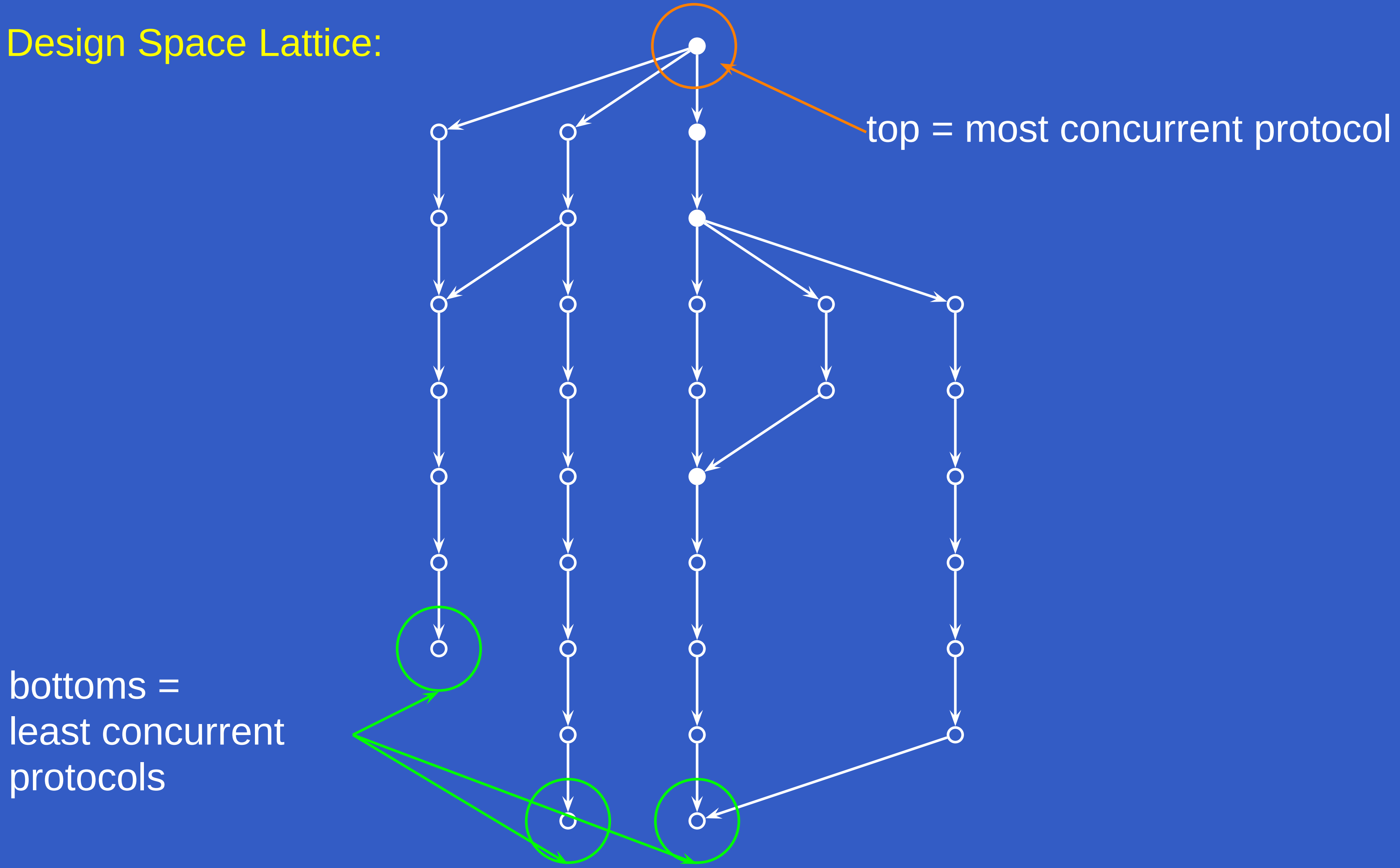
Design Space Lattice:



bottoms =
least concurrent
protocols

How Far Can You Move?

Design Space Lattice:

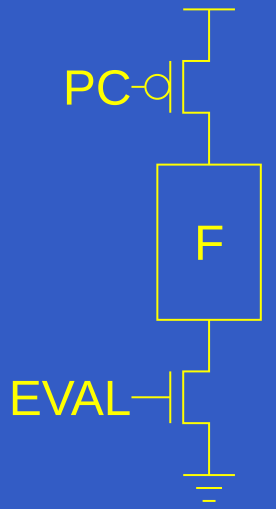


Deriving the Top of the Lattice

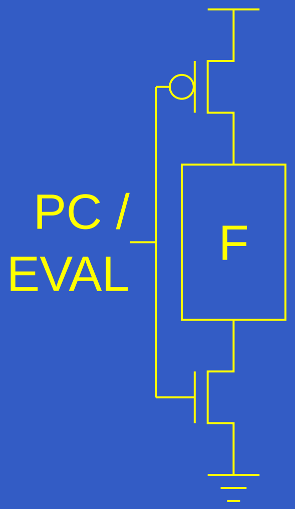
- How much concurrency can we have?
  - Natural limits based on correct (safe) operation of the pipeline implementation
  - Two dangers:
    - data overrun
    - reading stale data
- Goal: derive a graph which corresponds to the most concurrent protocol in the lattice

Deriving the Top of the Lattice

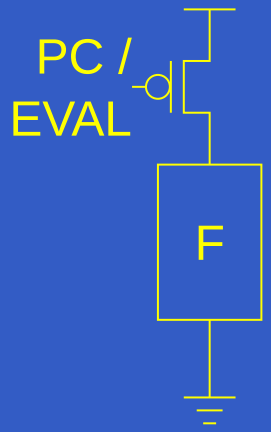
A different top for each logic design style:



Footed dynamic logic with separate controls



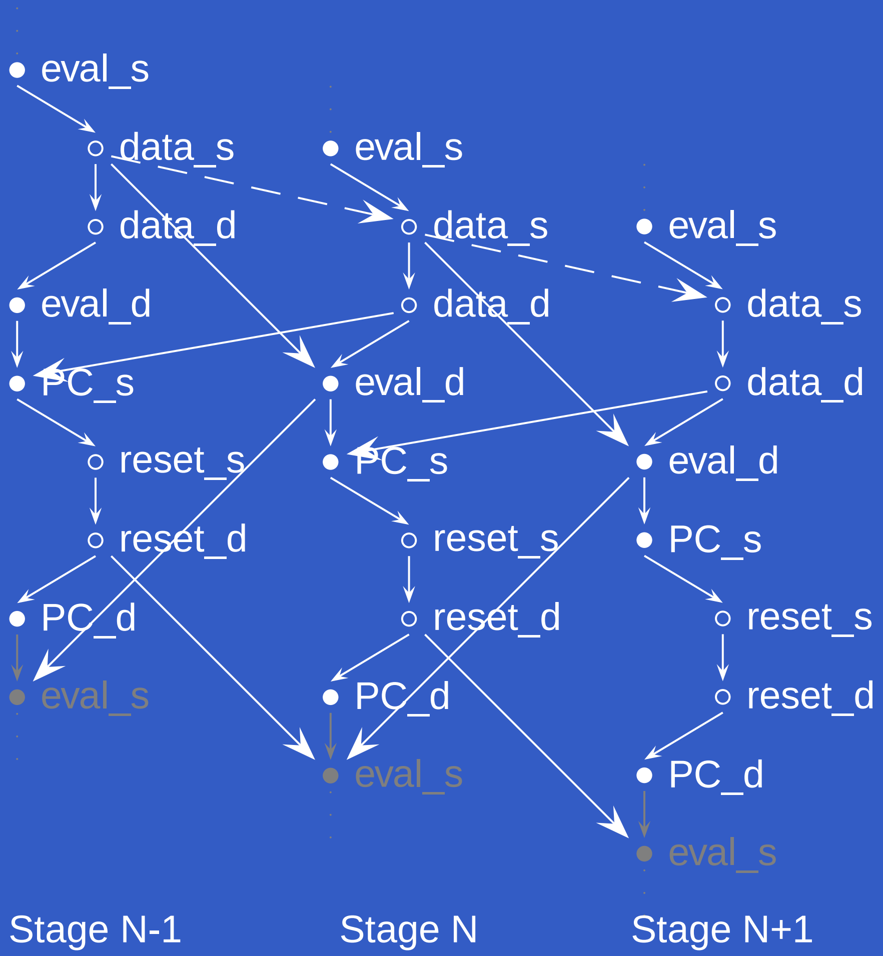
Footed dynamic logic with single control



Non-footed dynamic logic

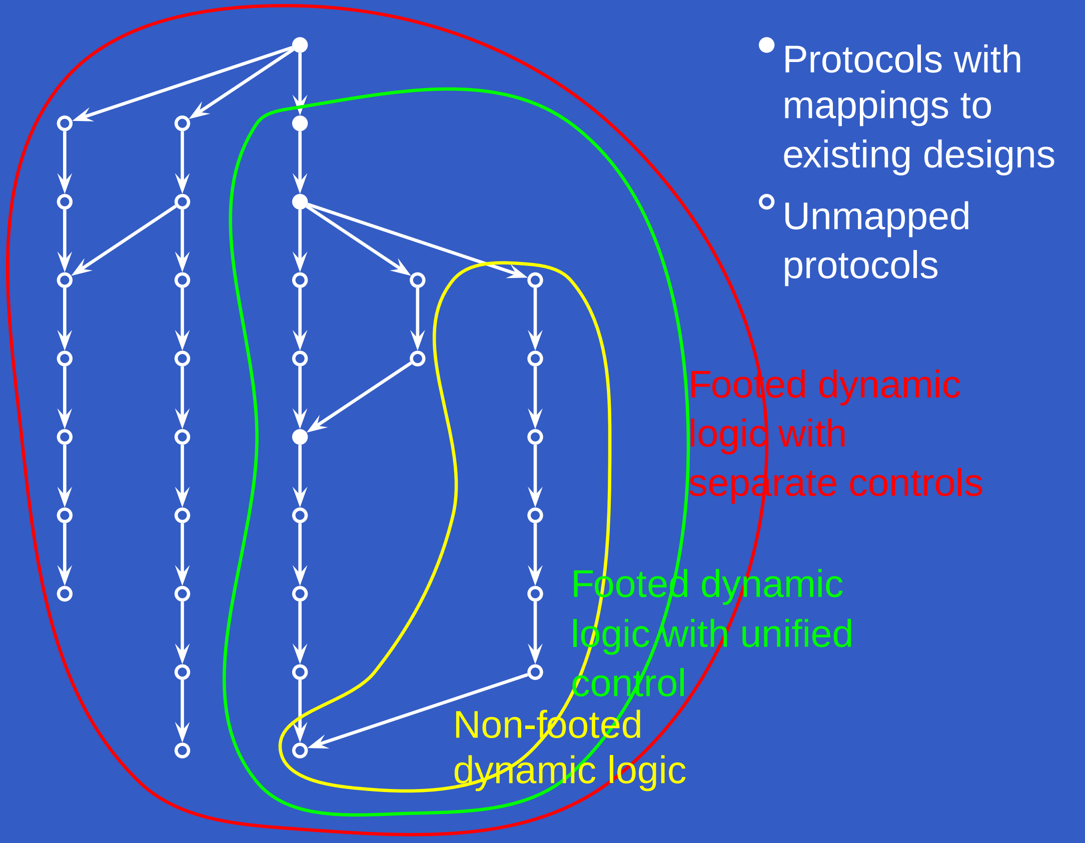
Deriving the Top of the Lattice

Most constrained protocol for pipelines using footed dynamic logic with separate controls:



Summary: Technical Approach

- Most concurrent protocol is derived from knowledge of the circuit and its interaction with data
- Entire design space can be generated from successive applications of transformation rules
- Prevention of deadlocks gives a lower bound to the design space



● Protocols with mappings to existing designs
○ Unmapped protocols
Footed dynamic logic with separate controls
Footed dynamic logic with unified control
Non-footed dynamic logic

Summary: Key Contributions

- Graph-based model of pipeline protocols: based on partial ordering of *data* and *circuit-level control* events
- Correct-by-construction transformation rules: for systematic exploration of design space
- A taxonomy of pipeline protocols: captured in a semi-lattice with well-defined top and bottom elements
- Handles several logic families:
  - dynamic
    - footed vs. non-footed
    - decoupled vs. unified control
  - static

Related Work and Future Work

- Related work
  - Furber et al. [1996] and Lines [1998]: family of pipeline circuits with different interleavings of handshake signals
  - Blunno et al. [Async 04]: hierarchy of asynchronous control circuits using static logic
  - None provided well-defined design space boundary, or formal, systematic way for design space exploration
- Future work
  - Synthesize abstract protocol to hardware