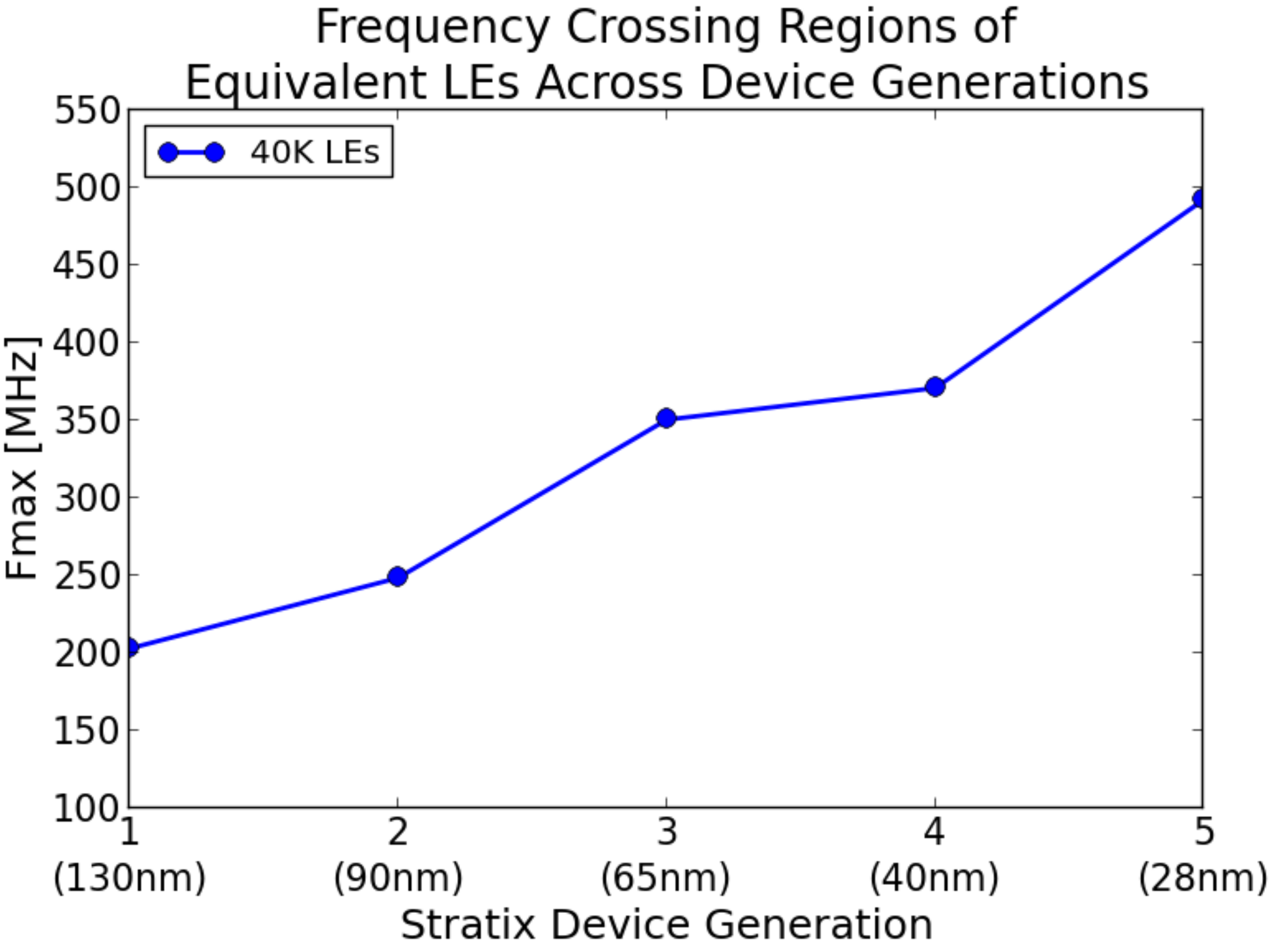
Quantifying the Cost and Benefit of Latency Insensitive Communication on FPGAs

Kevin E. Murray and Vaughn Betz

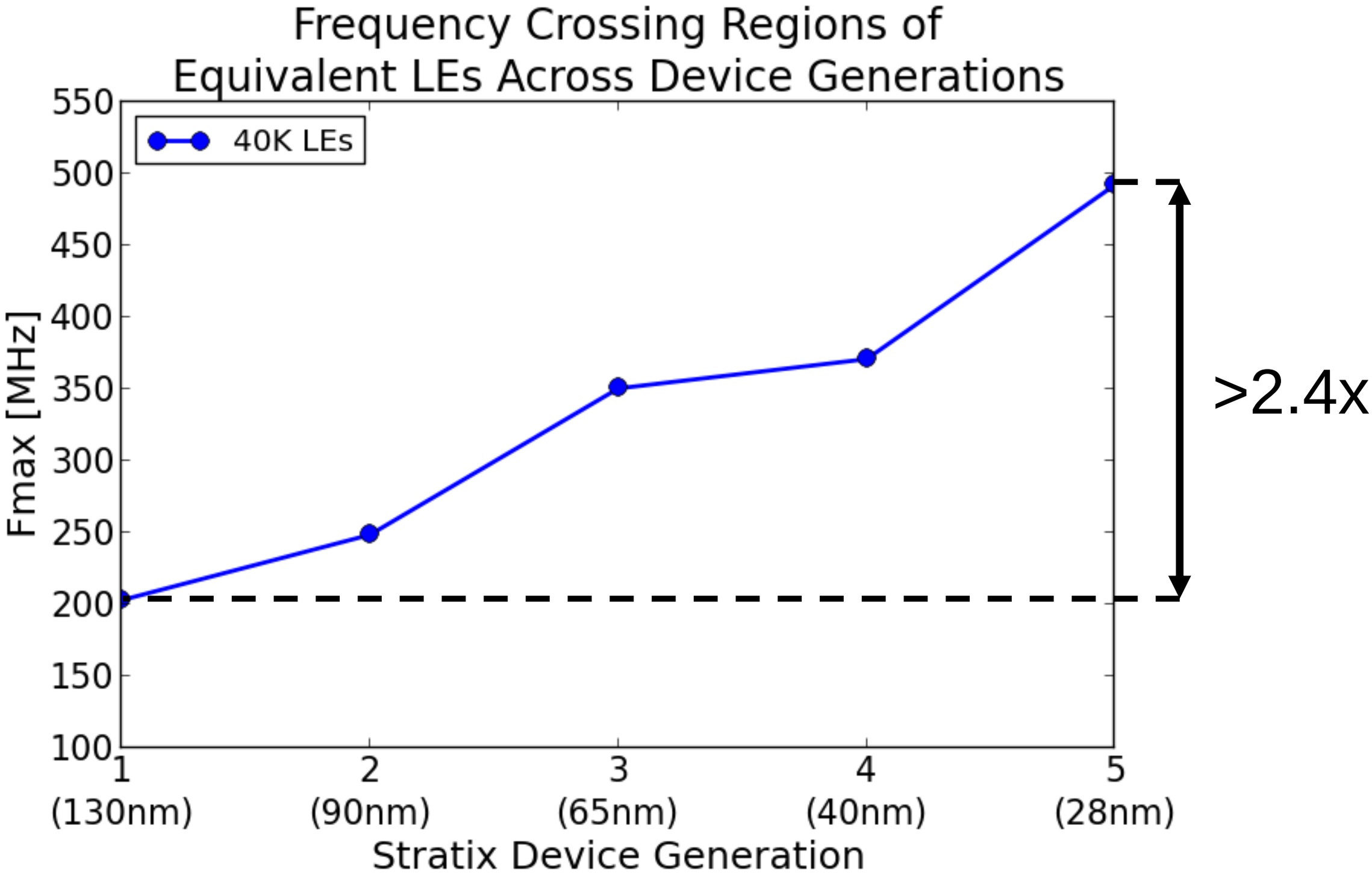
UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE & ENGINEERING

Motivation

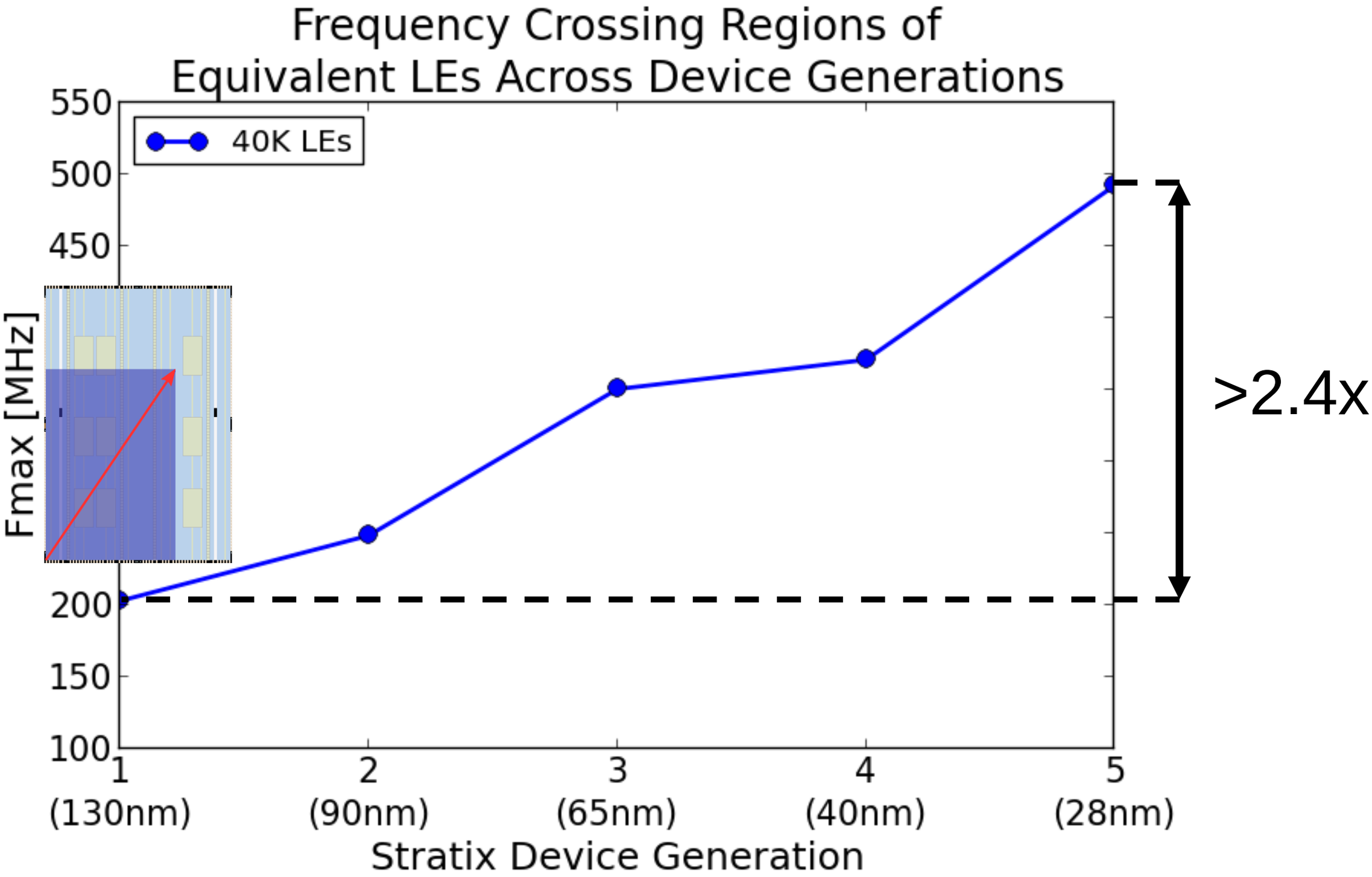
Local Communication Speed Improving



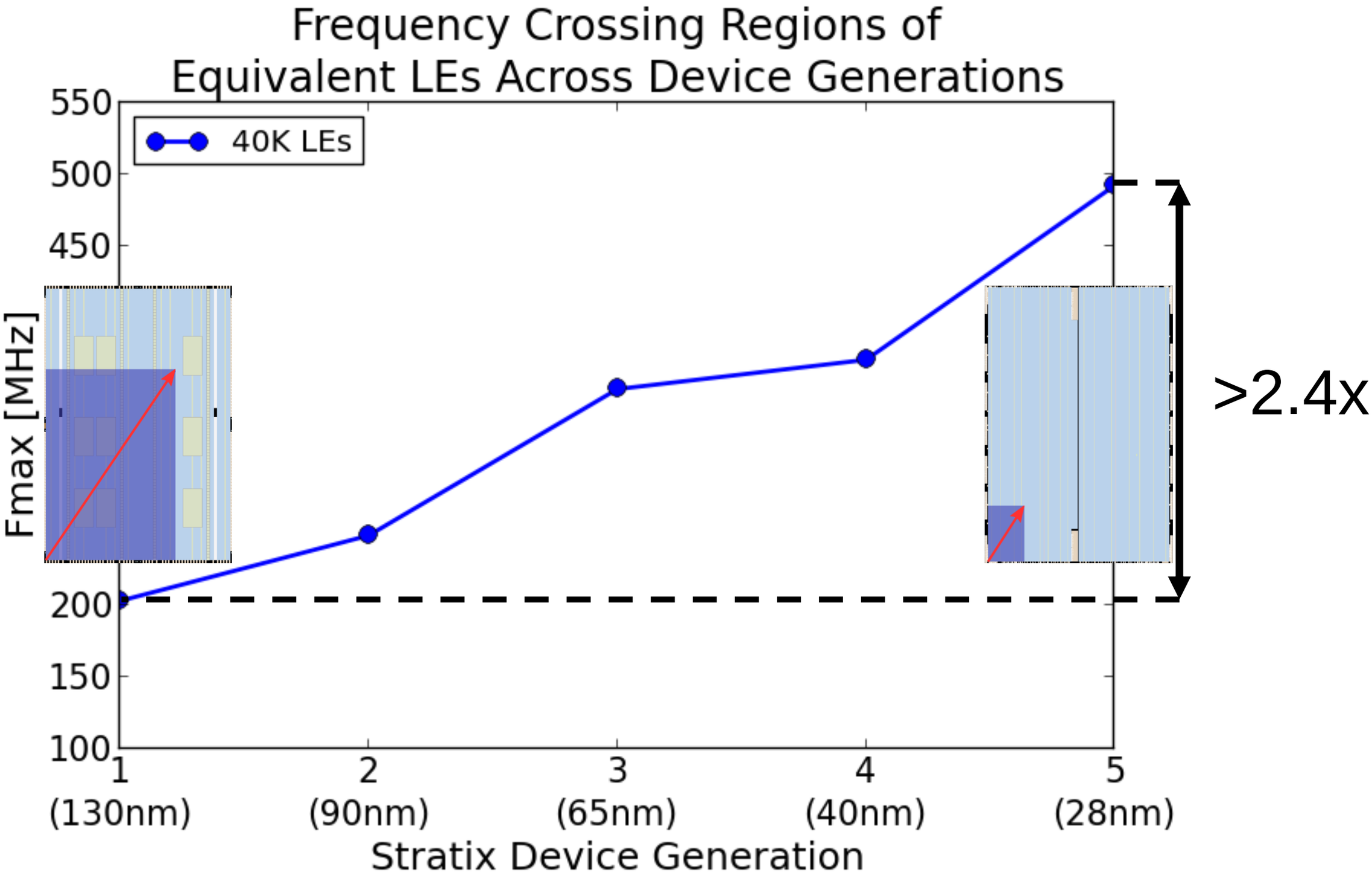
Local Communication Speed Improving



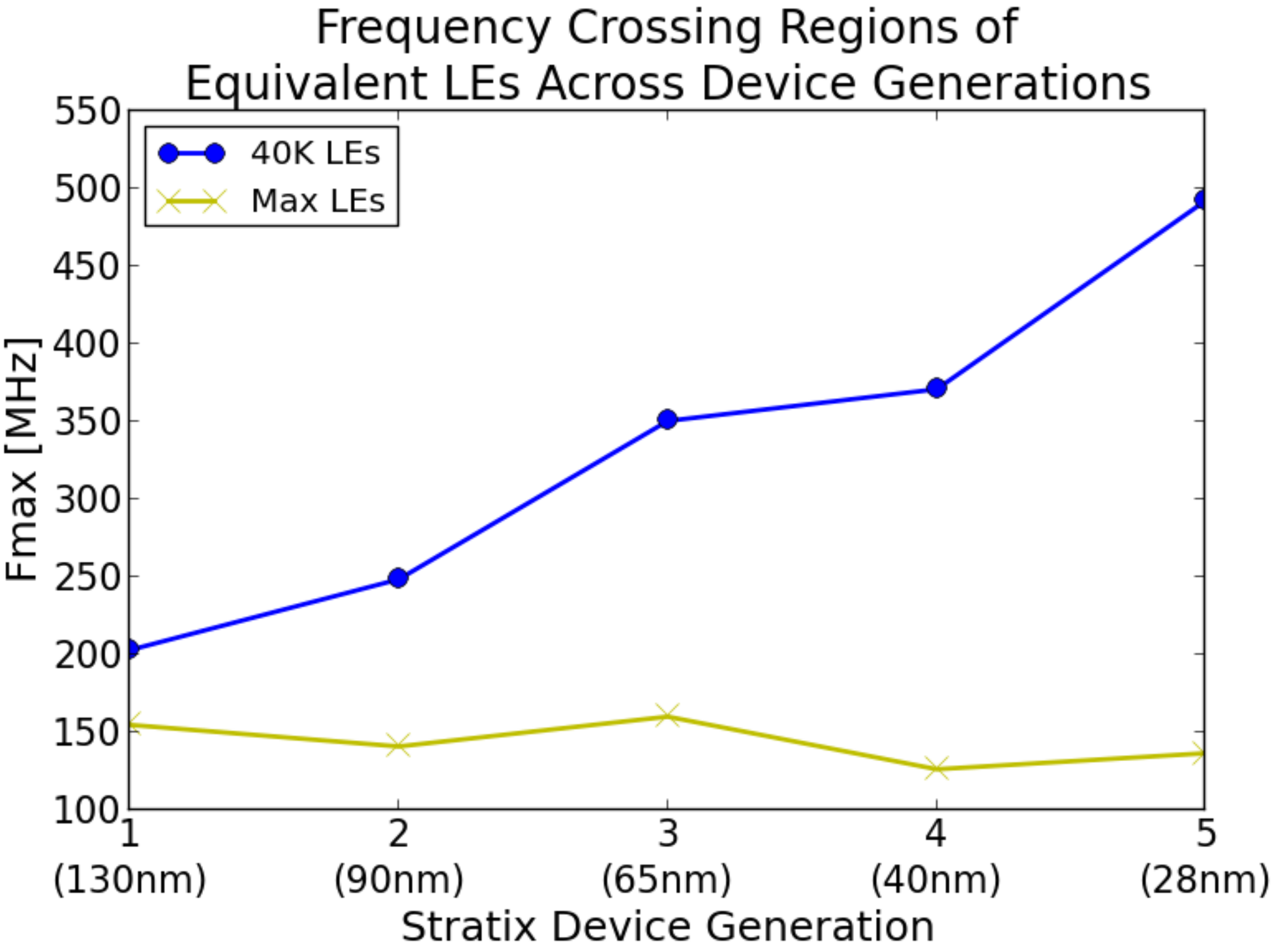
Local Communication Speed Improving



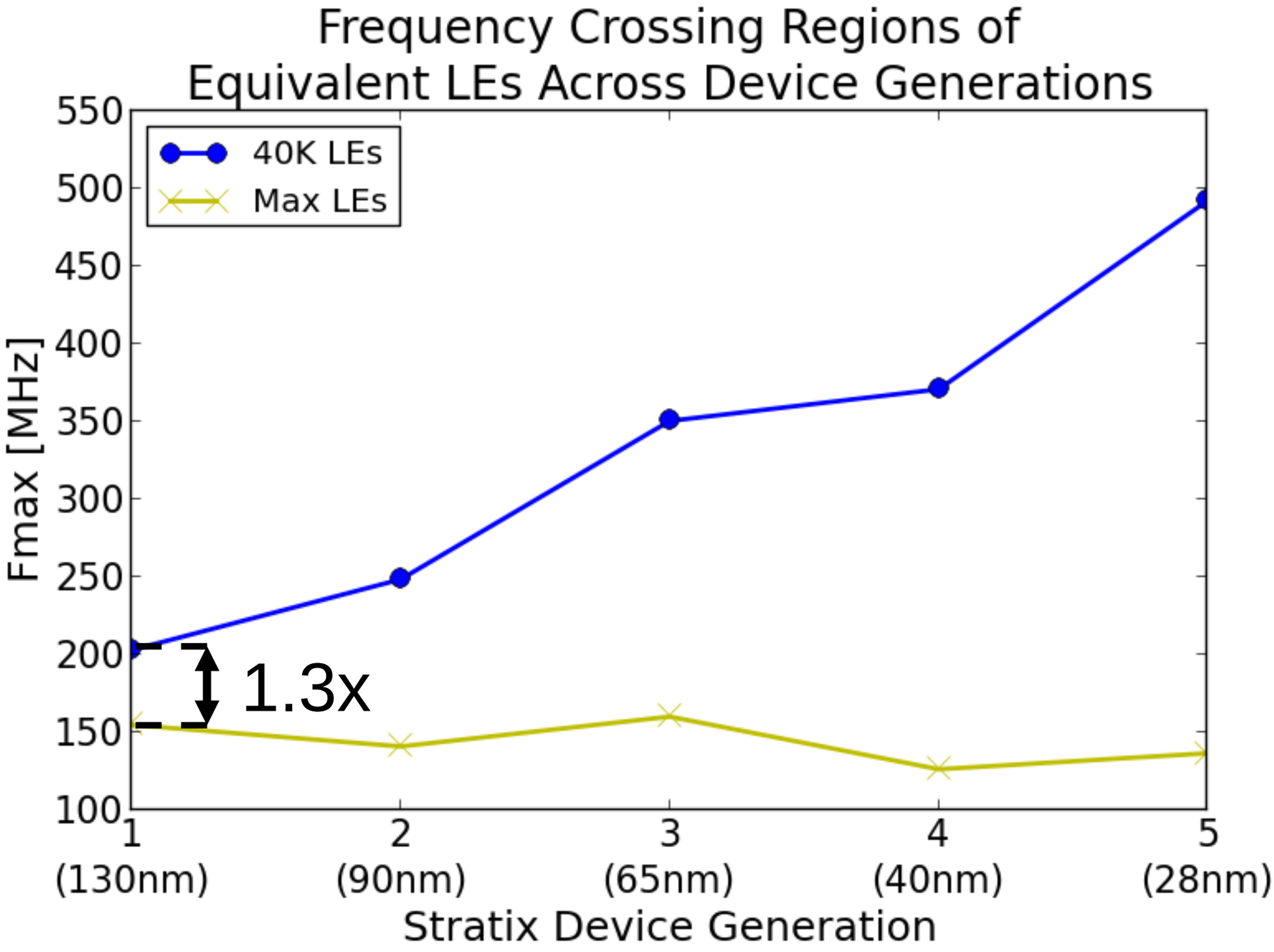
Local Communication Speed Improving



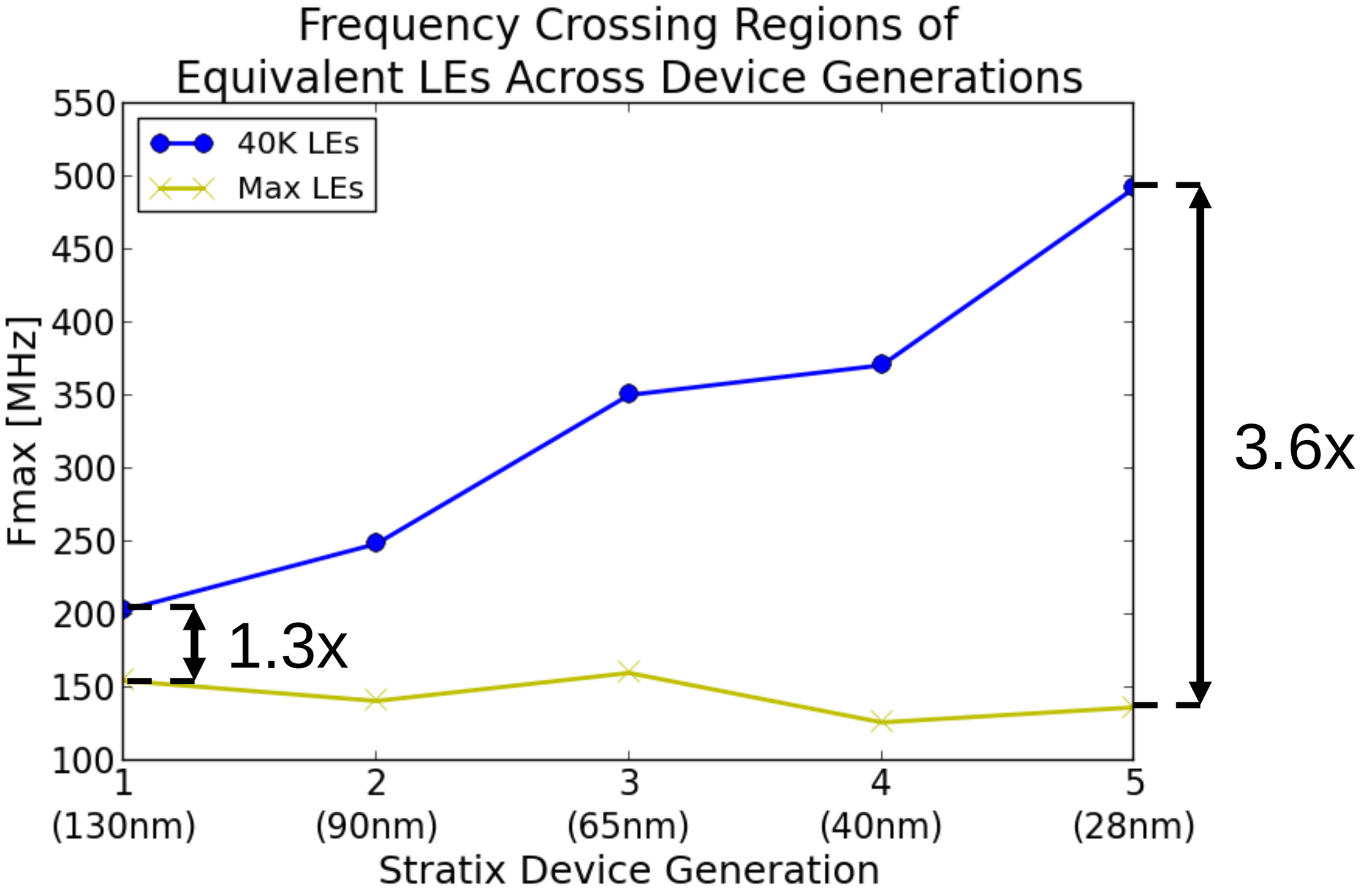
Global Communication Speed Not Improving



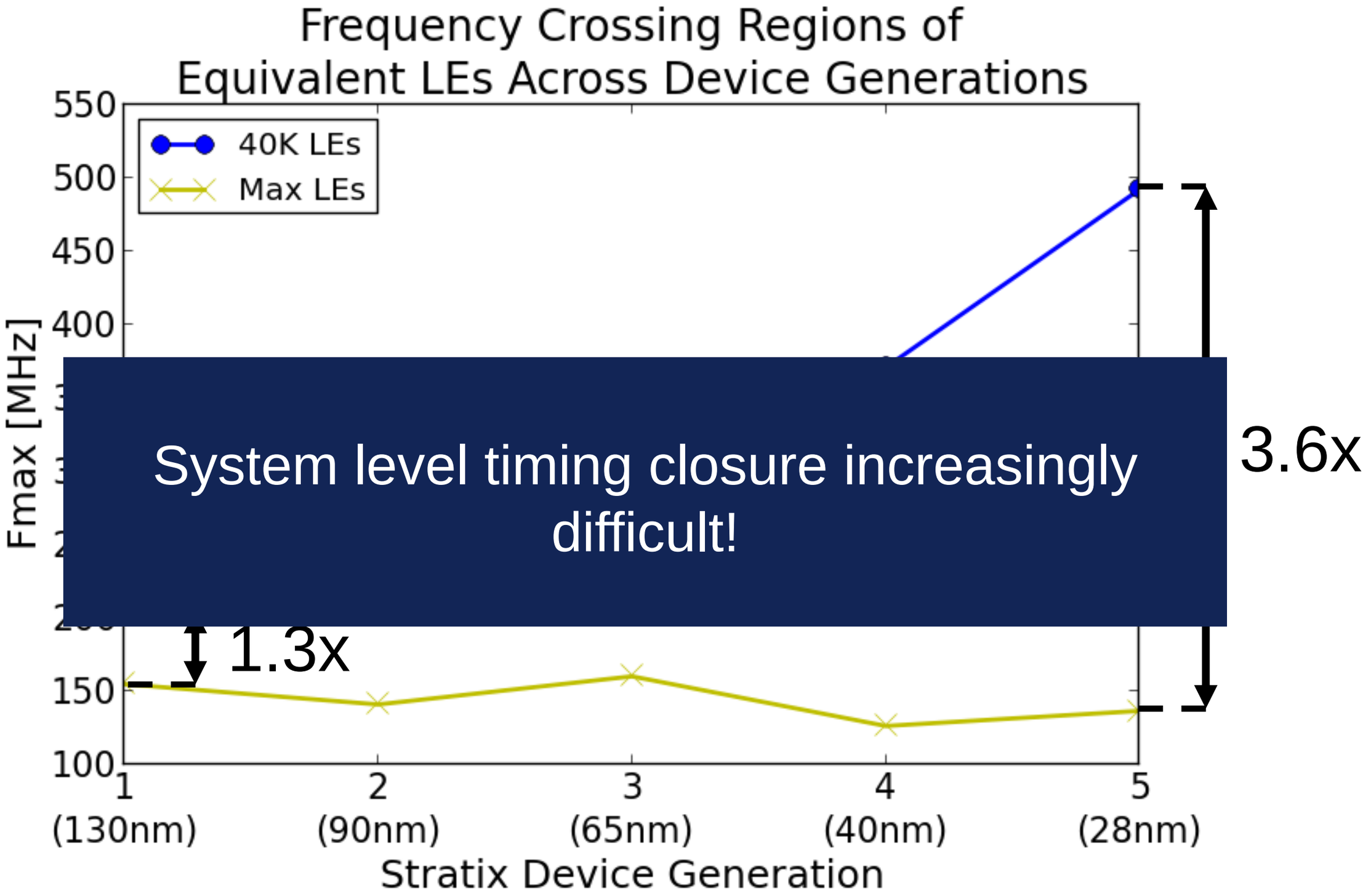
Global Communication Speed Not Improving



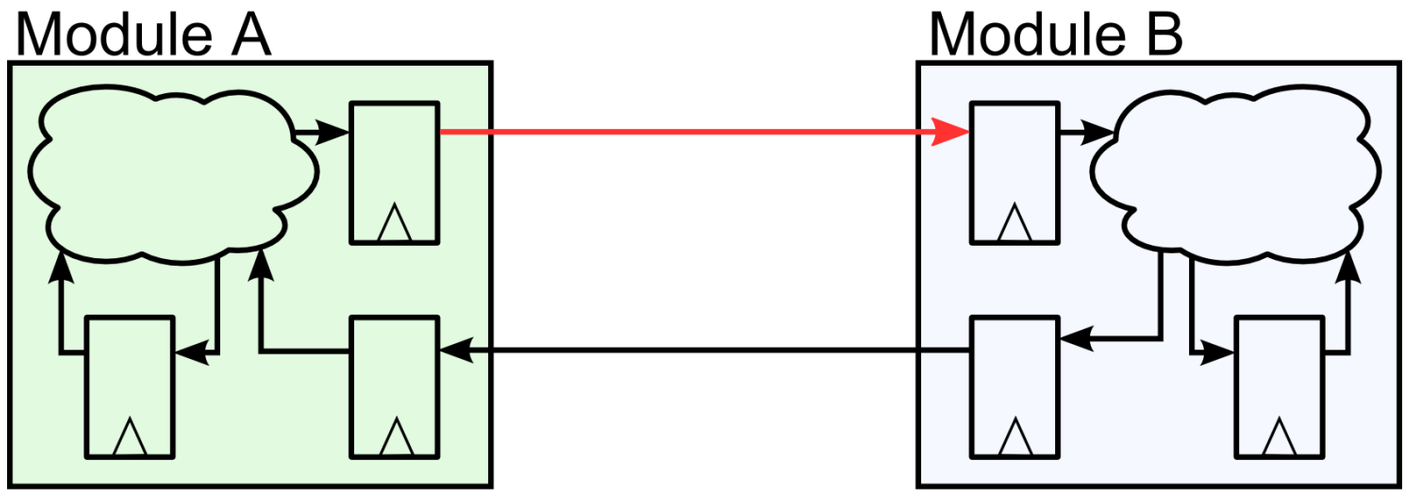
Global Communication Speed Not Improving



Global Communication Speed Not Improving

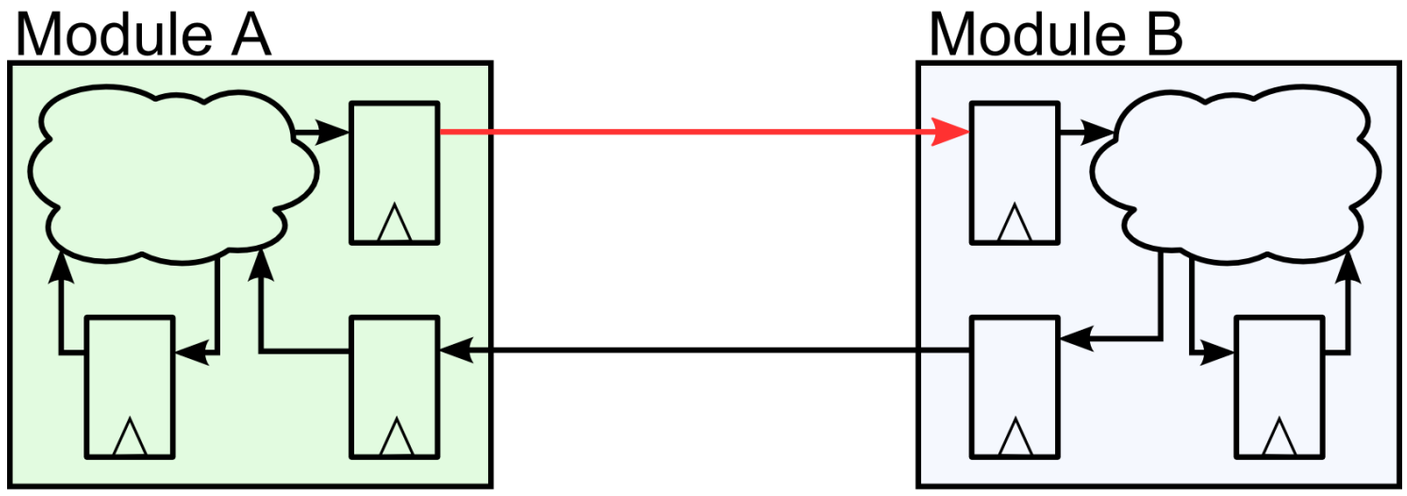


System Level Timing Closure Issues

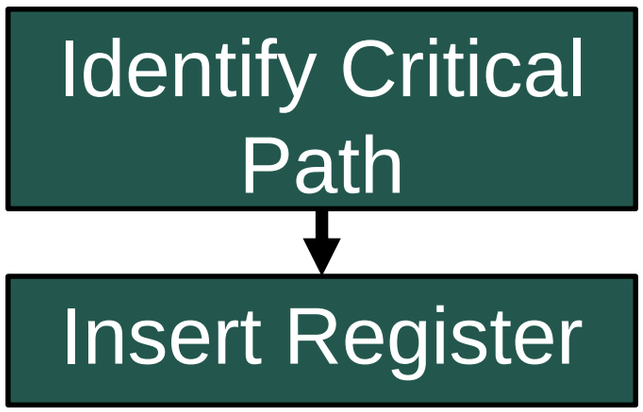


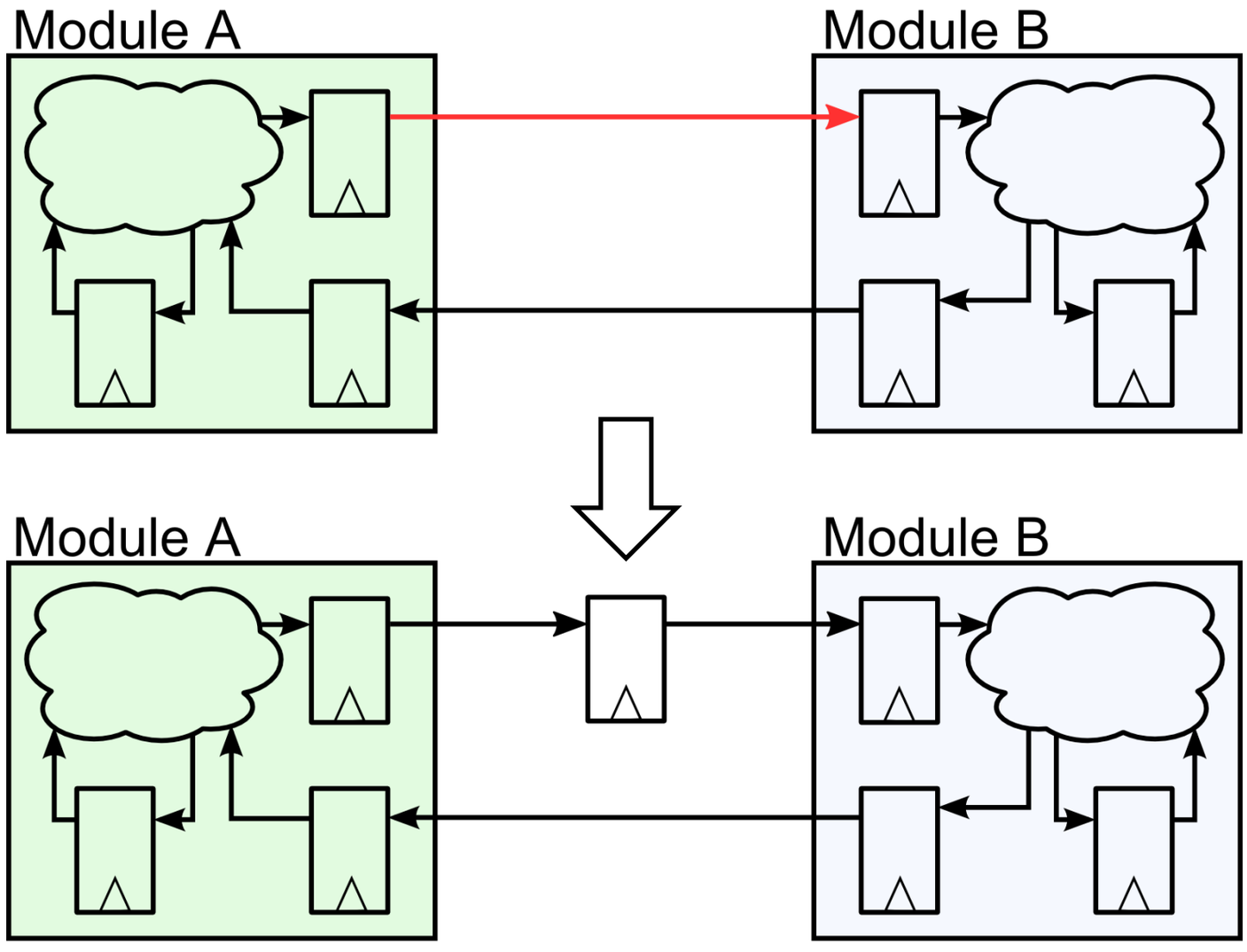
System Level Timing Closure Issues

Identify Critical Path



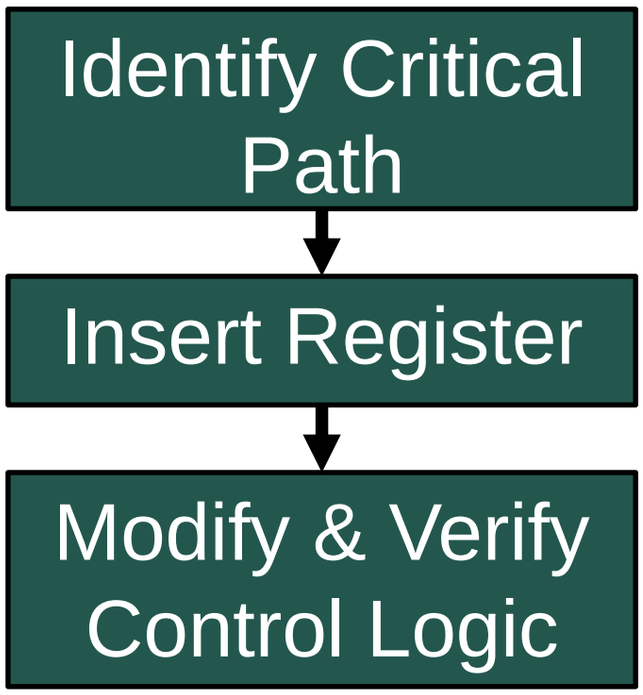
System Level Timing Closure Issues

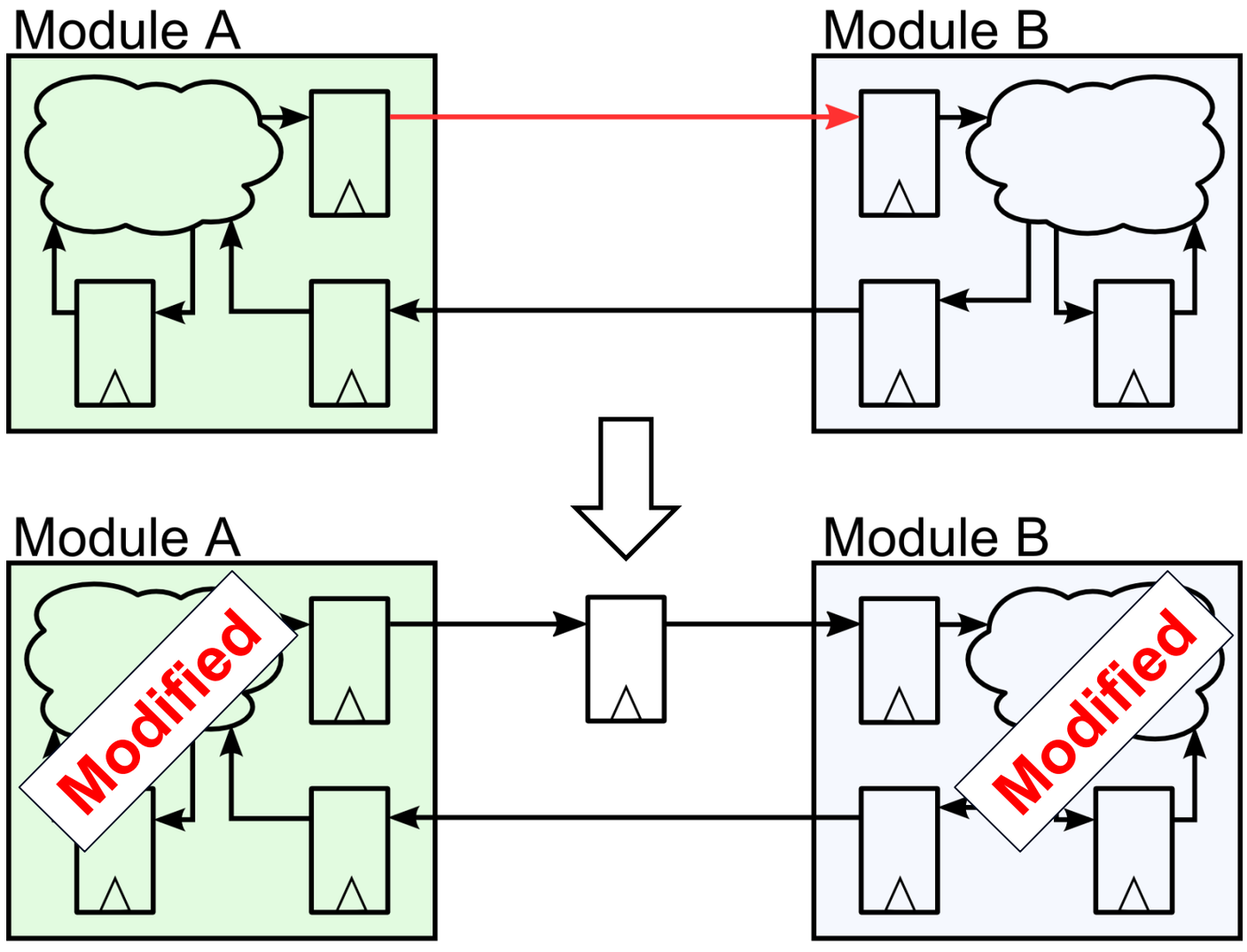




System Level Timing Closure Issues

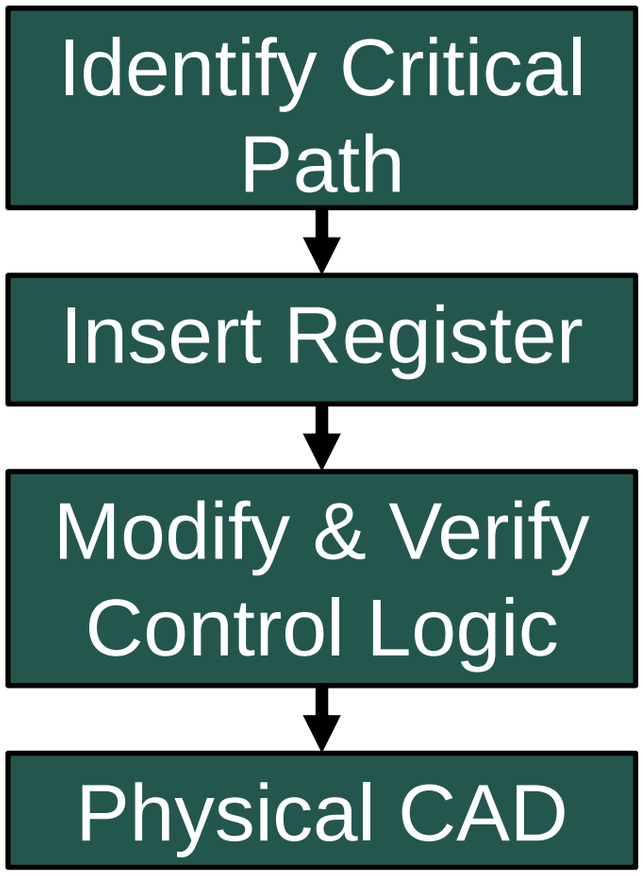
- RTL changes are error prone

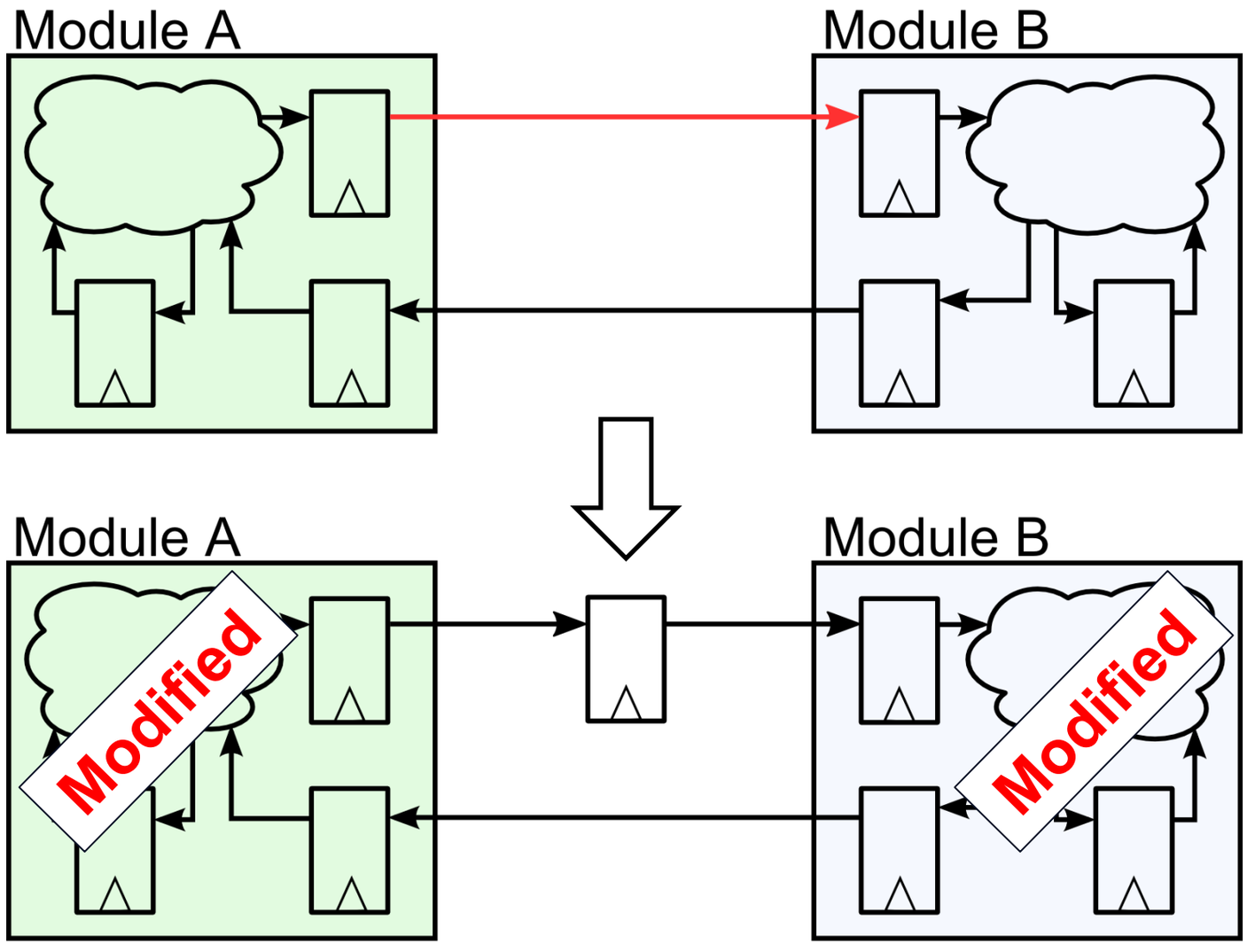




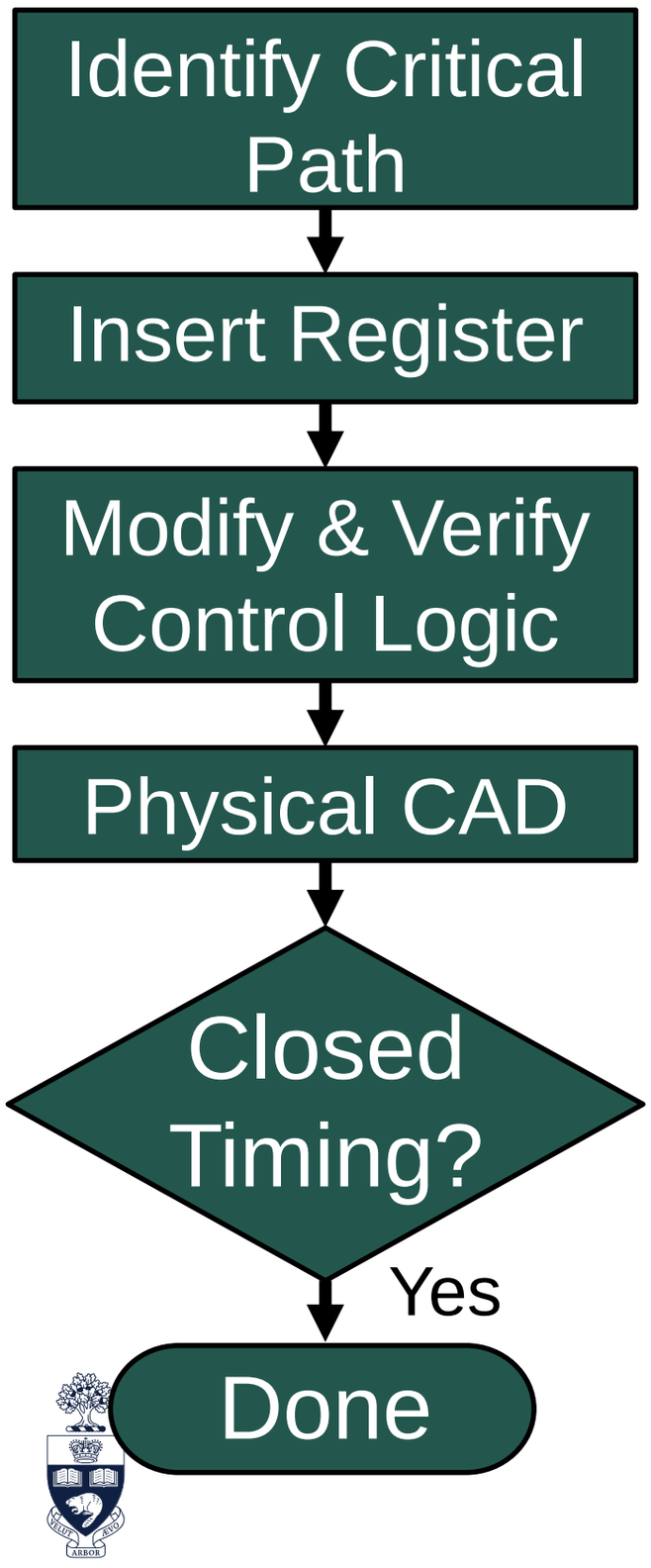
System Level Timing Closure Issues

- RTL changes are error prone
- Re-compile can take hours or days

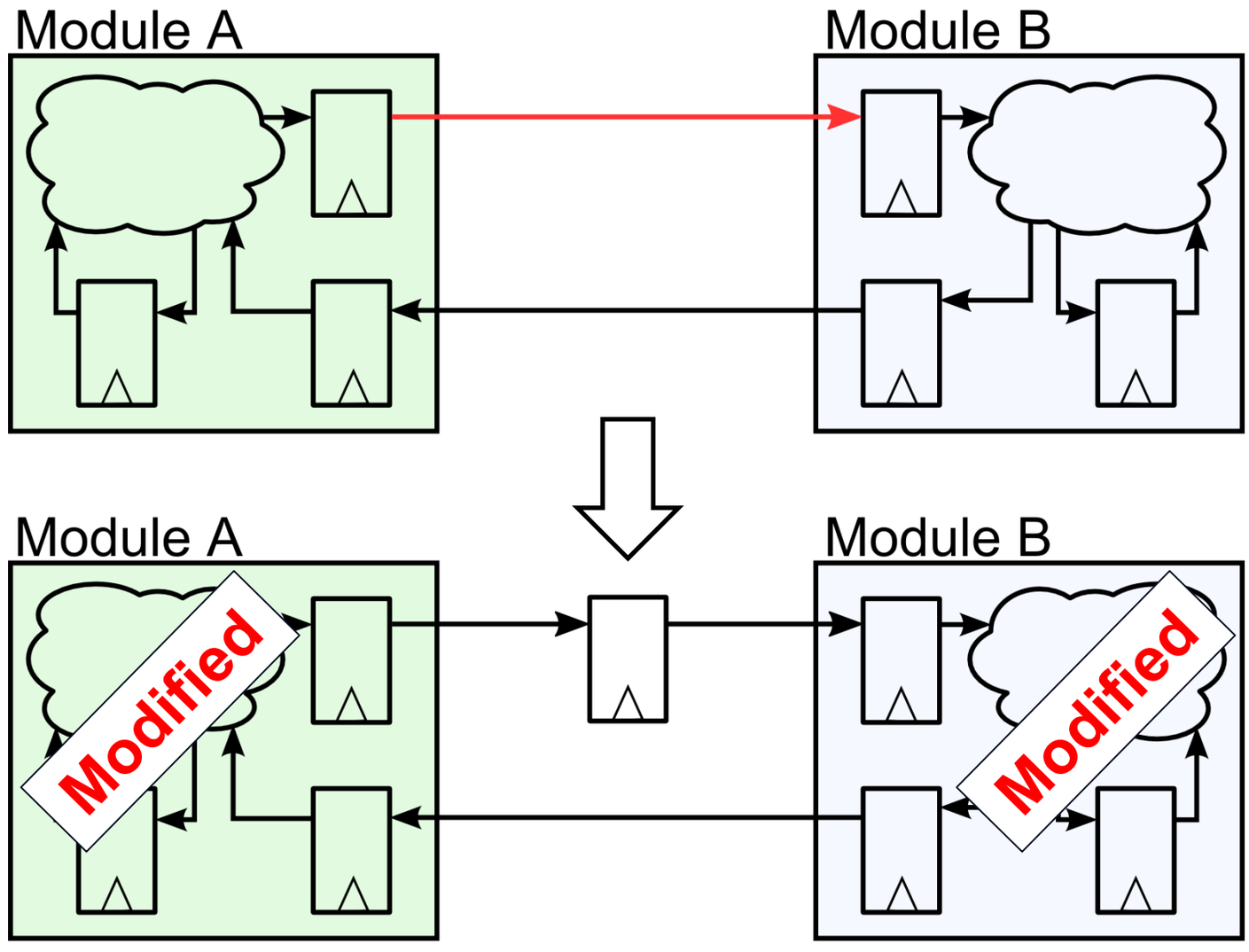




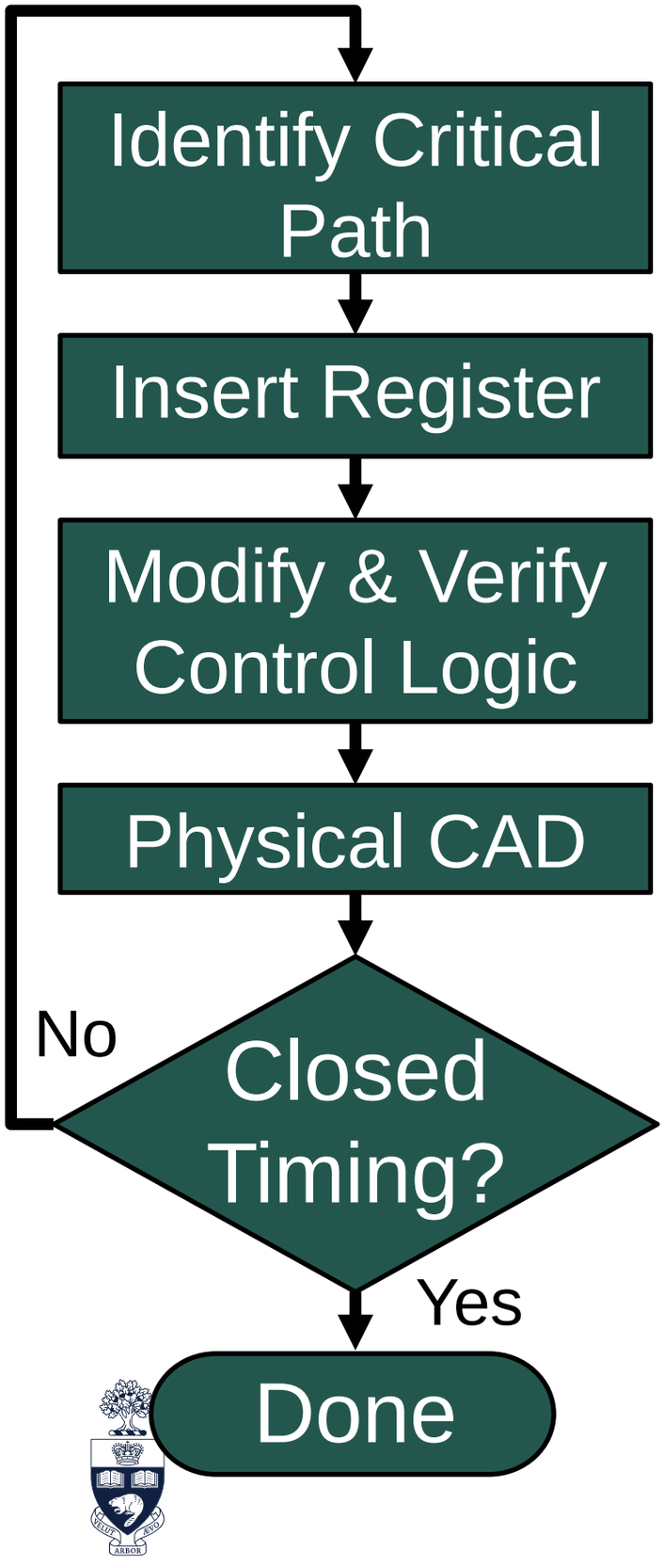
System Level Timing Closure Issues



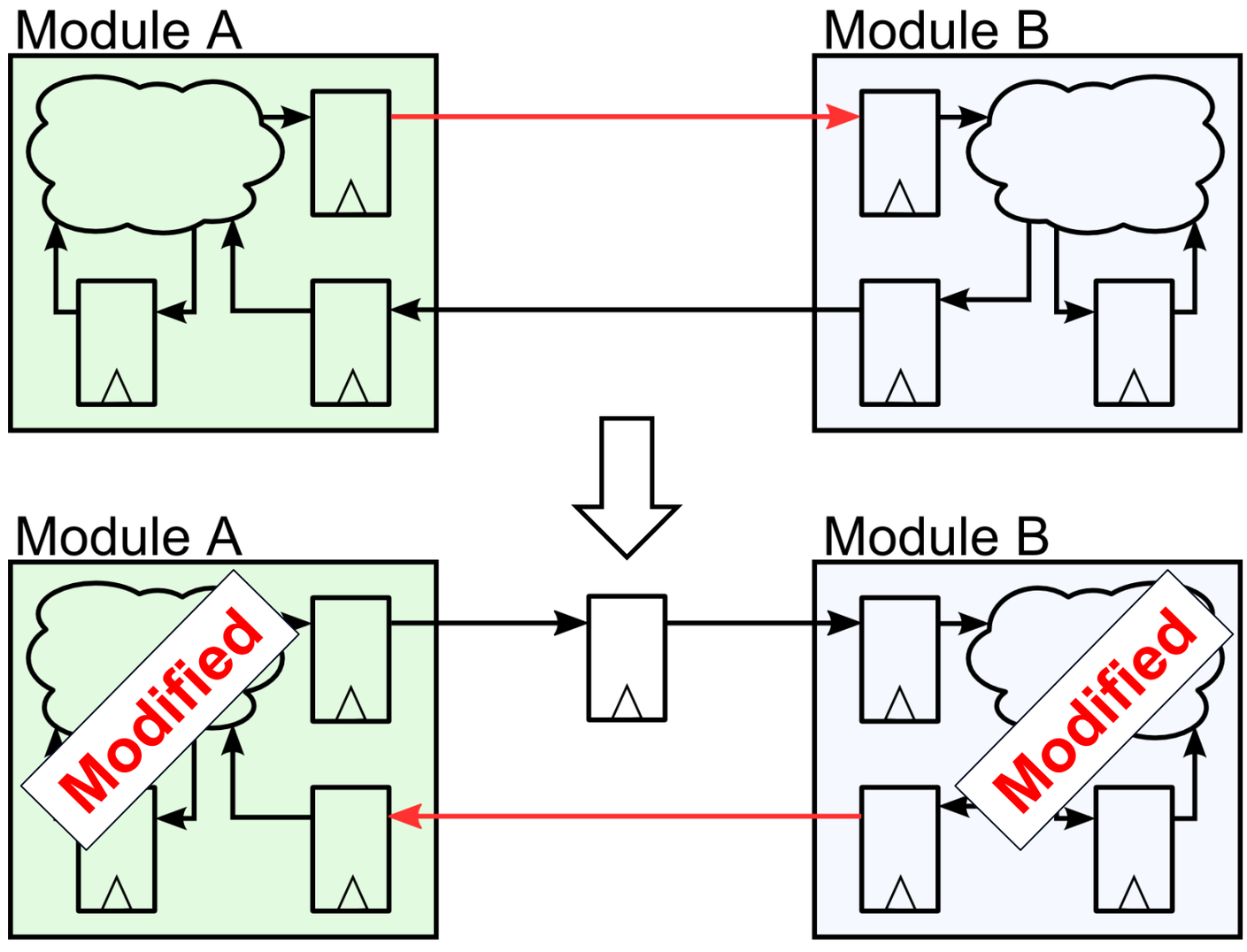
- RTL changes are error prone
- Re-compile can take hours or days



System Level Timing Closure Issues



- RTL changes are error prone
- Re-compile can take hours or days
- No guarantee of convergence



Key Problem & Potential Solutions

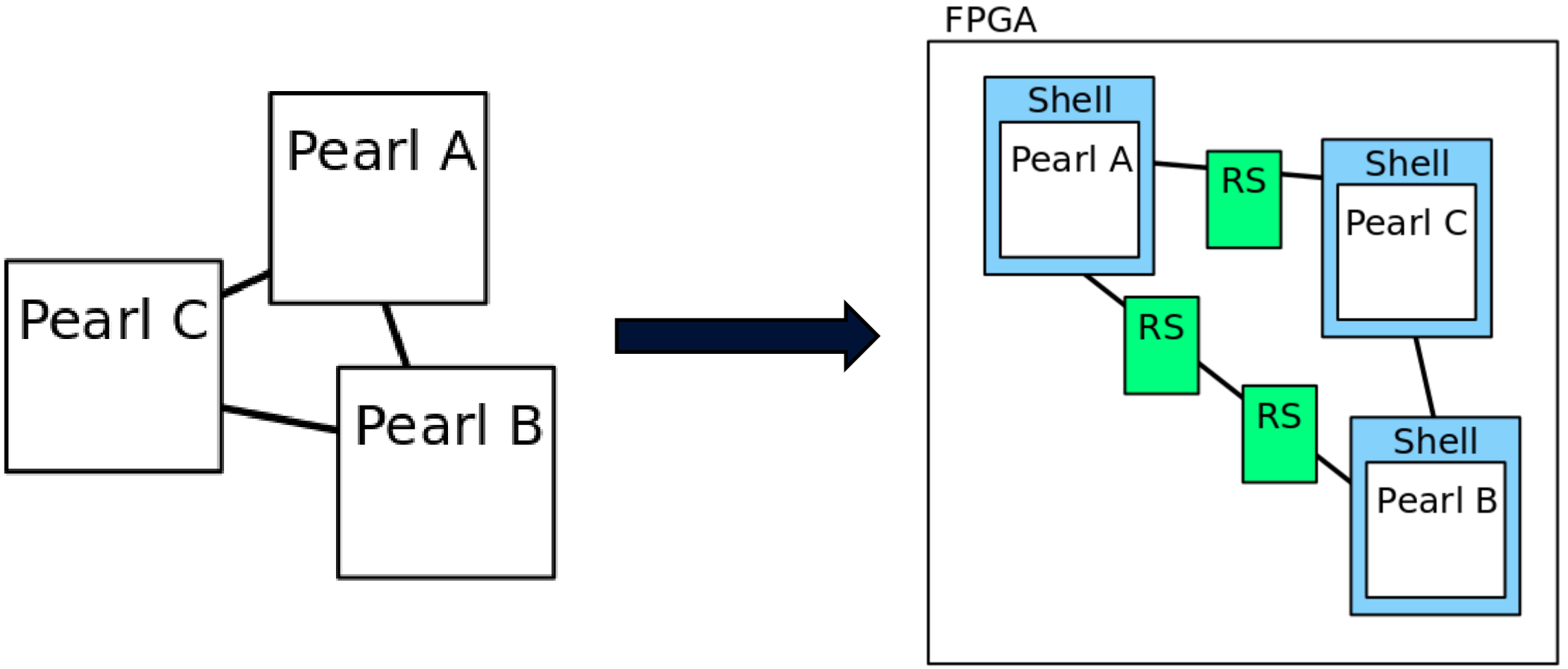
- Limited by *Synchronous Assumption*:
  - Computation and communication occur in a single clock cycle (if not pipelined)
  - Reasonable when local-global speed gap was small, but not when large

- Many different proposed design schemes:
  - Over-pipelining
  - Asynchronous
  - Globally Asynchronous Locally Synchronous (GALS)
  - **Latency Insensitive**

What is Latency Insensitive Design?

Latency Insensitive System Implementation

- **Key Idea:** Make sub-modules insensitive to their communication latency
- Create a LI module by placing a designer's synchronous module (*Pearl*) in a wrapper (*Shell*), which is also synchronous
- Use Relay Stations (*RS*) to pipeline interconnect
- Deadlock free and applicable to (nearly) any synchronous module [1]



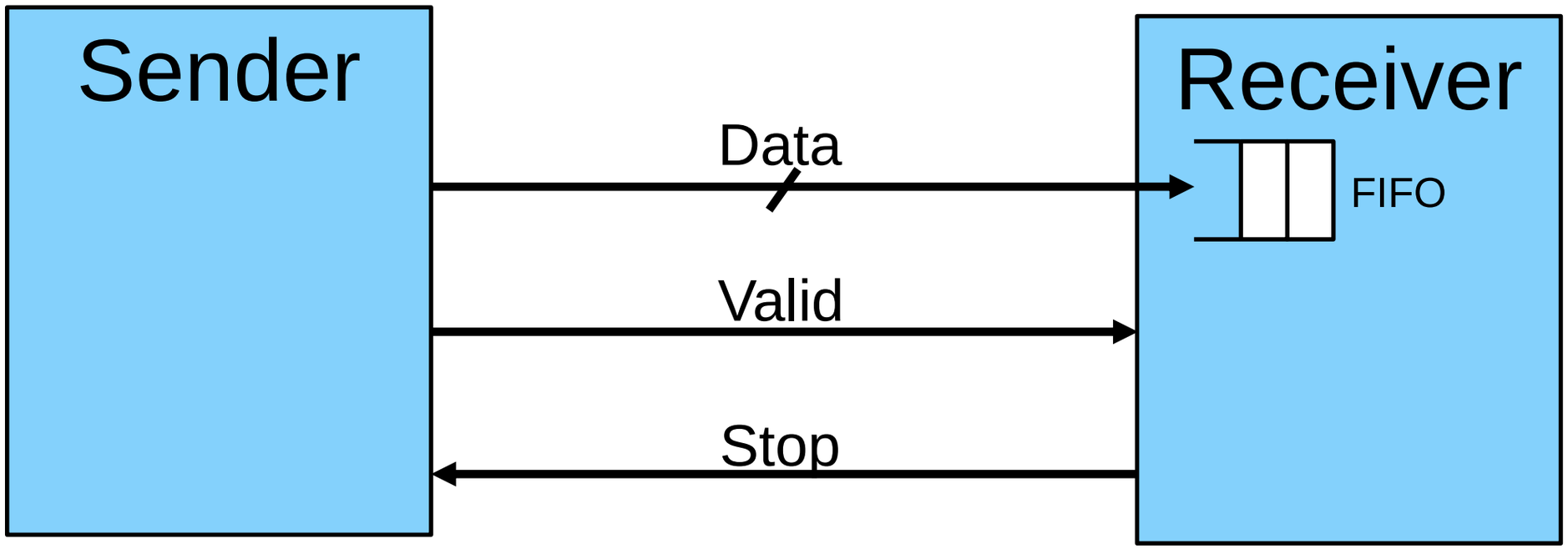
Logical System

LI Implementation

[1] Carloni et. al, "Theory of Latency-Insensitive Design", TCAD, 2001

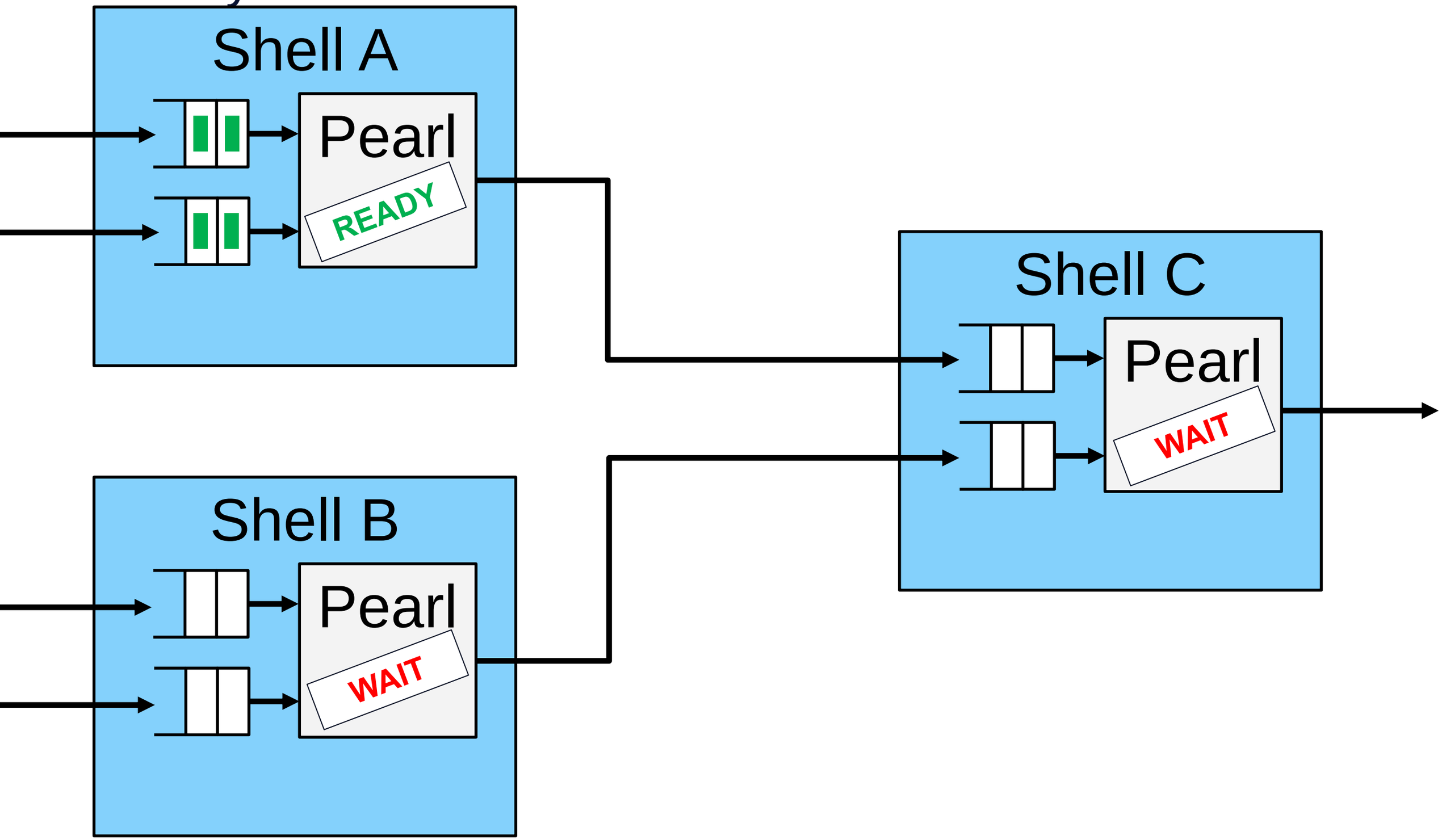
Latency Insensitive Communication Protocol

- Tag each module port with ‘Valid’ and ‘Stop’ bits
- Pearl is paused until all inputs are ‘valid’

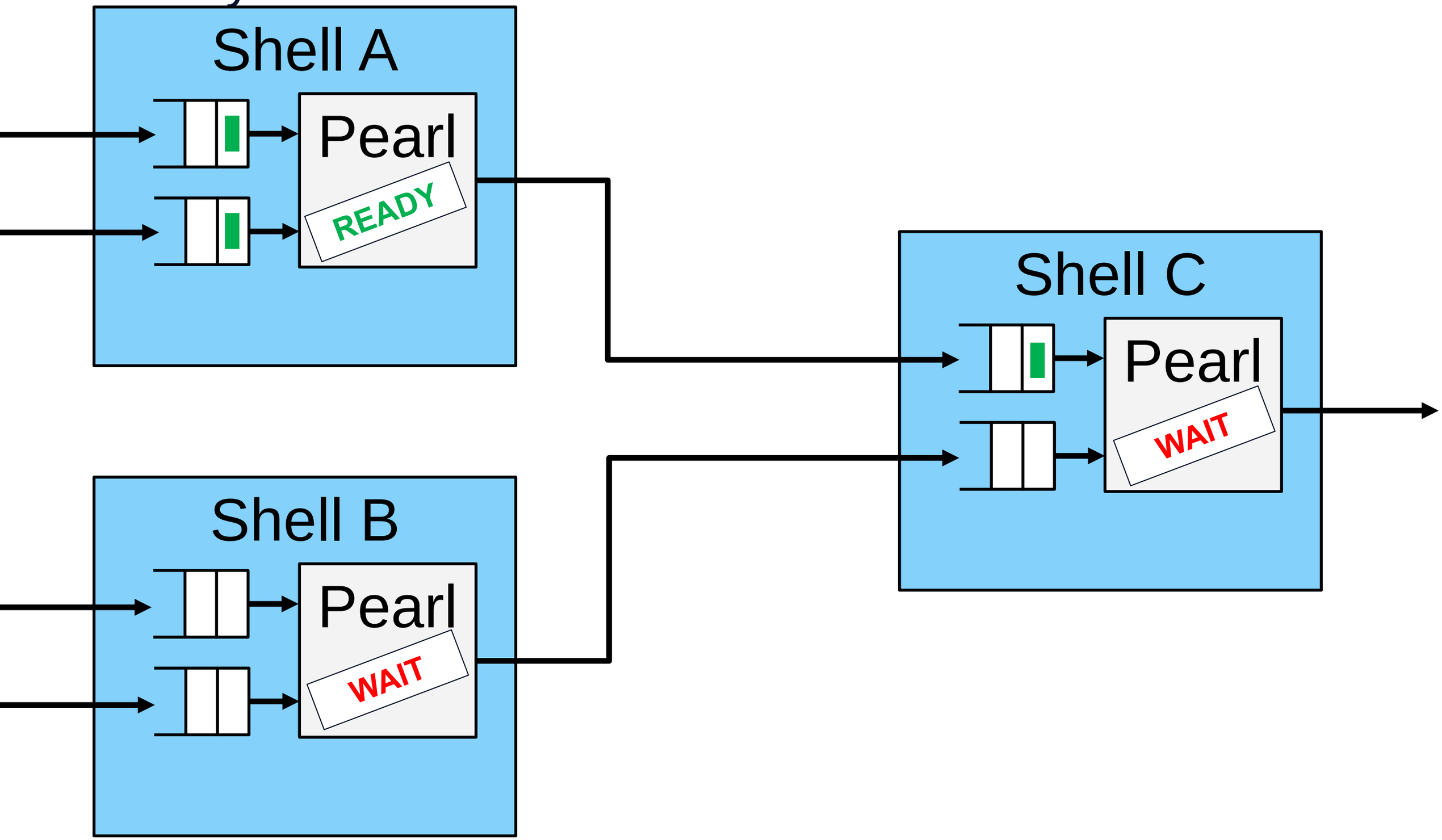


- ‘Stop’ signal provides back-pressure to prevent FIFO overflow

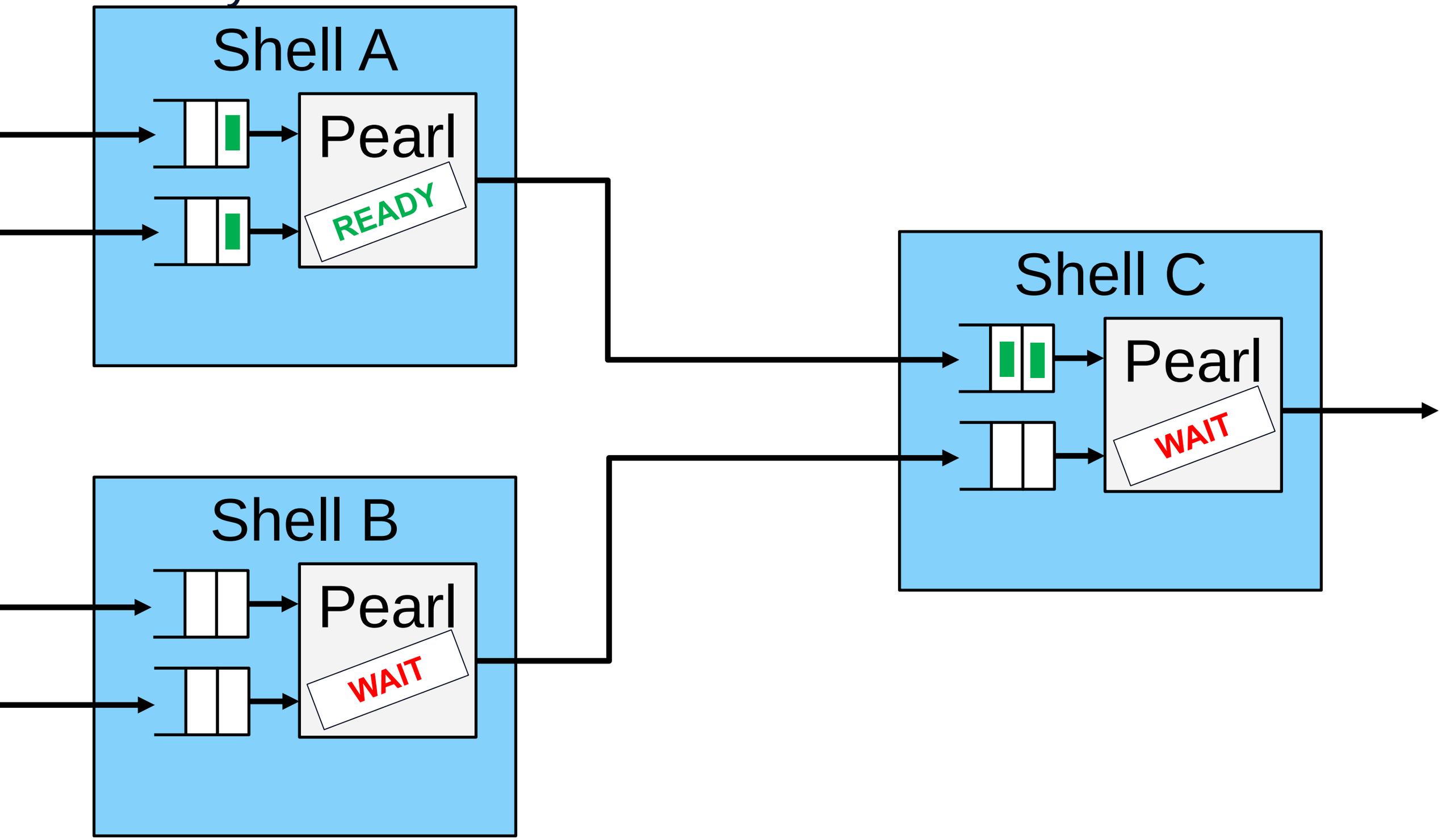
Latency Insensitive Communication Protocol



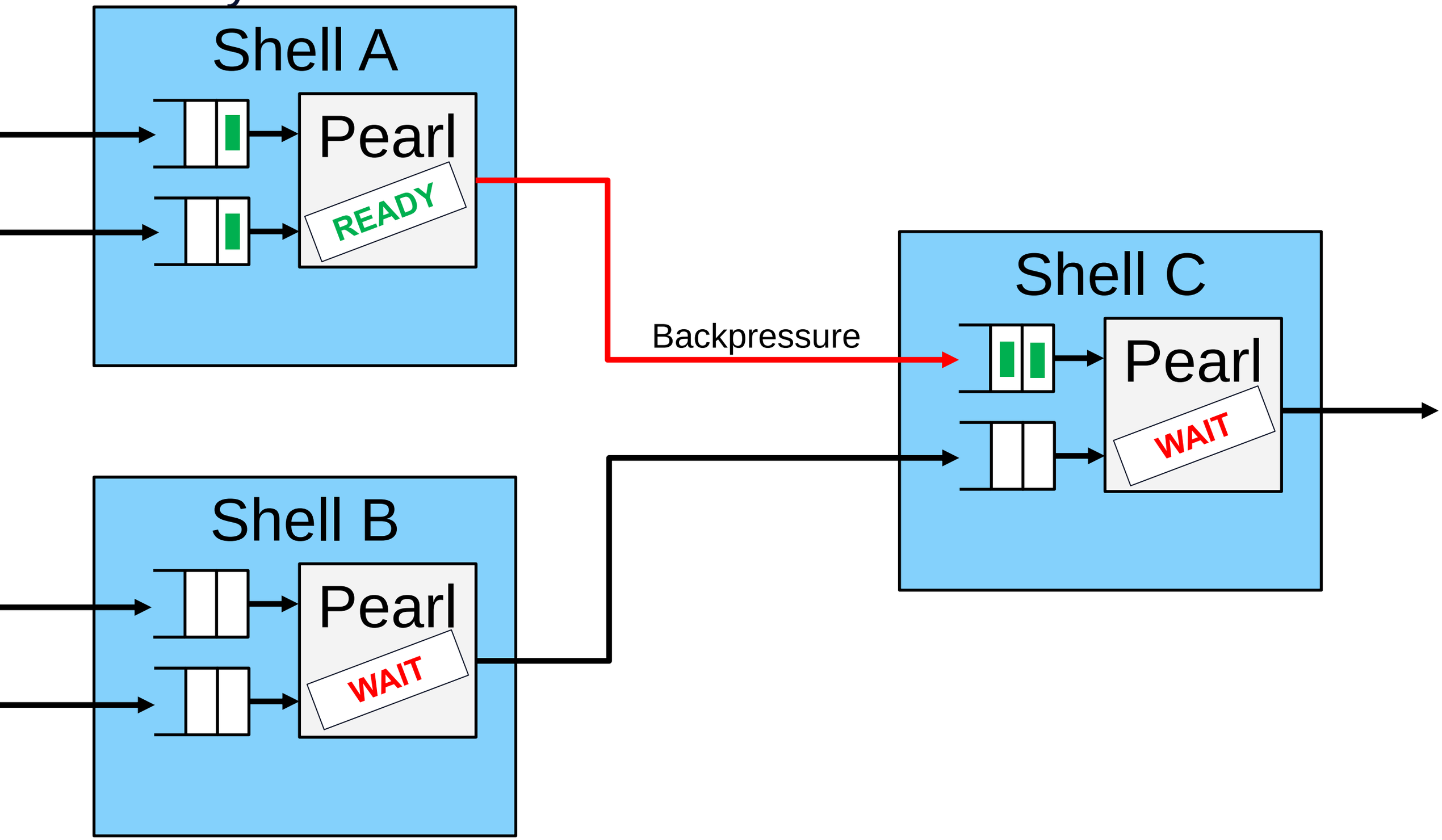
Latency Insensitive Communication Protocol



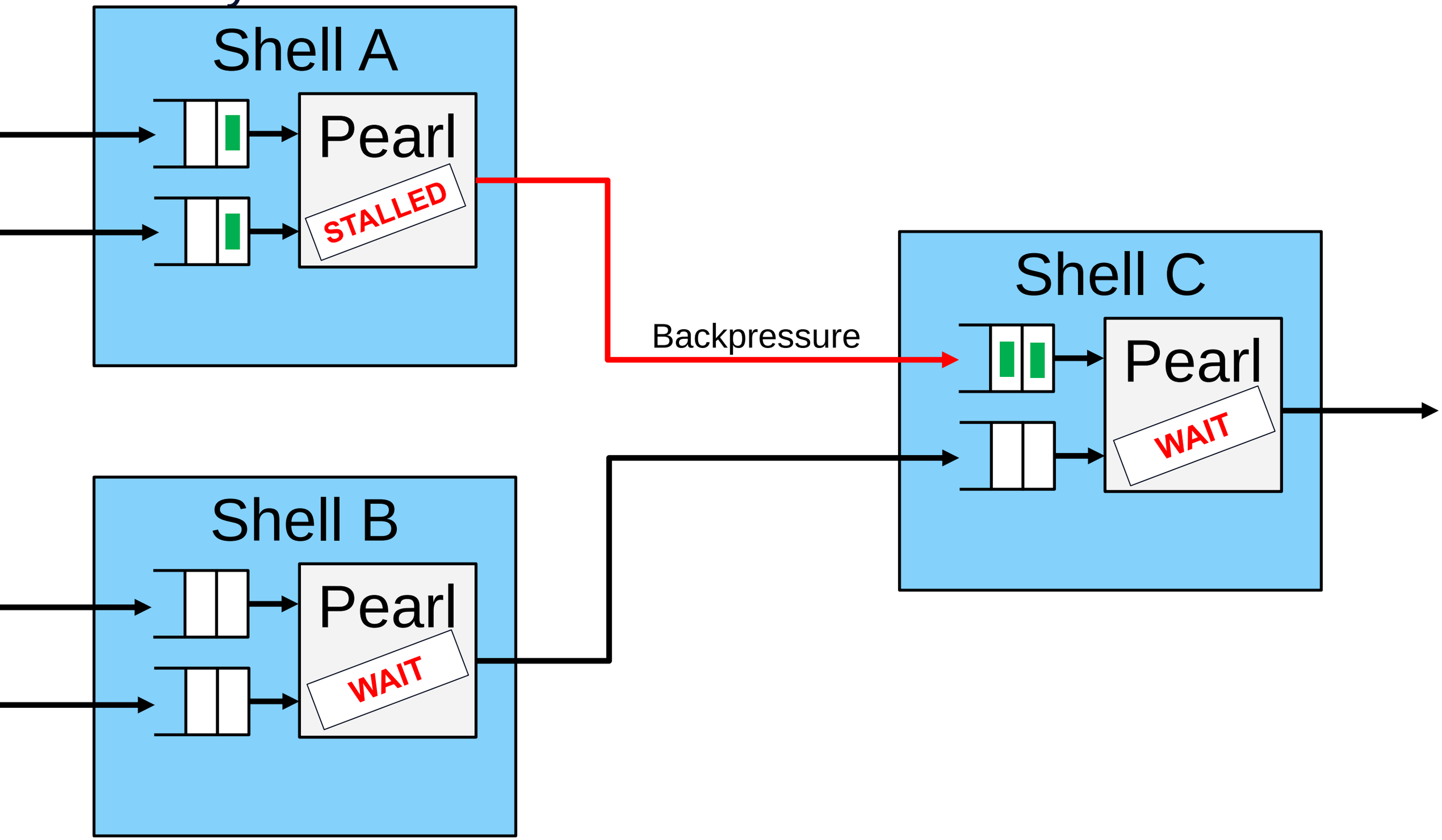
Latency Insensitive Communication Protocol



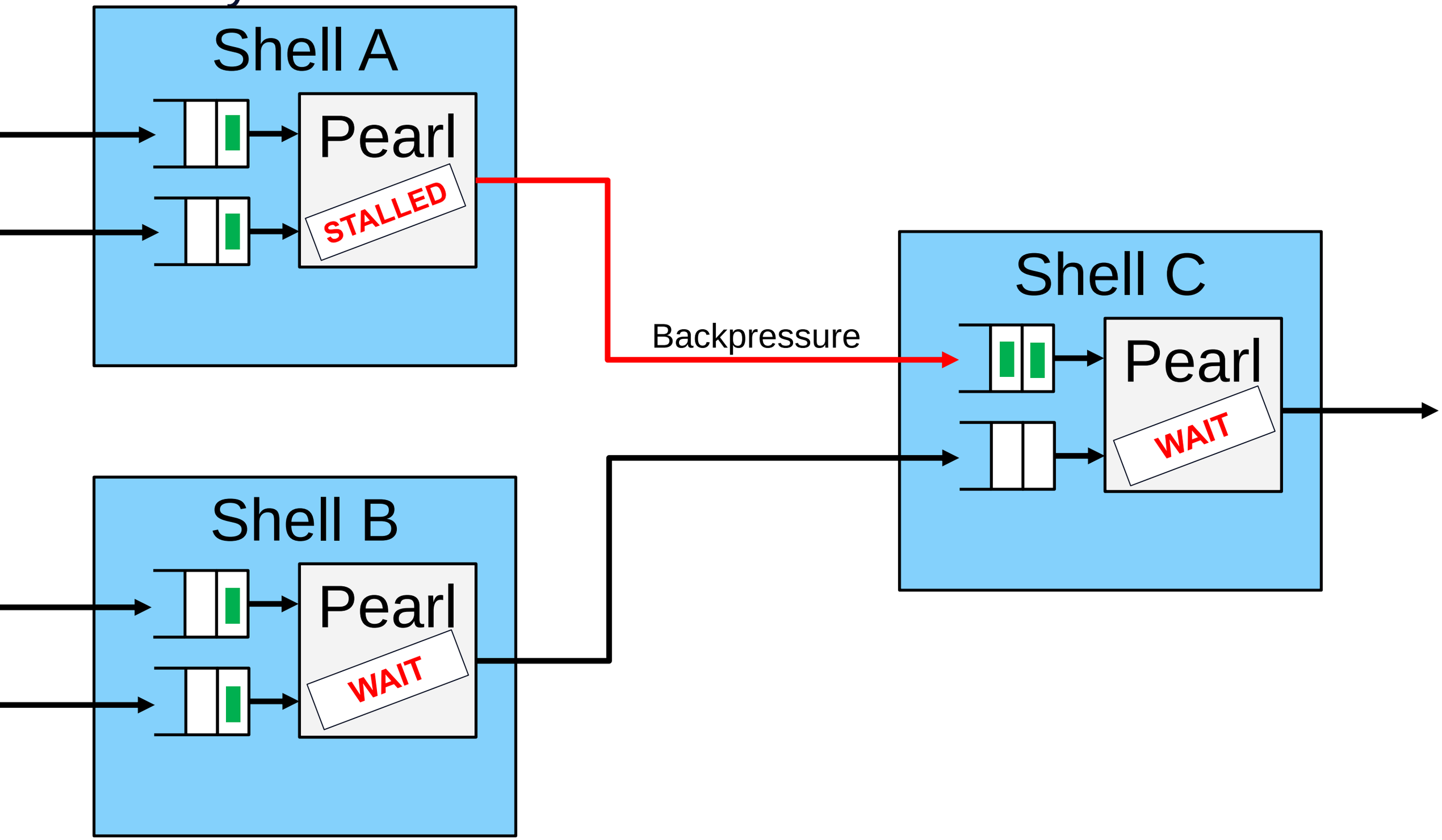
Latency Insensitive Communication Protocol



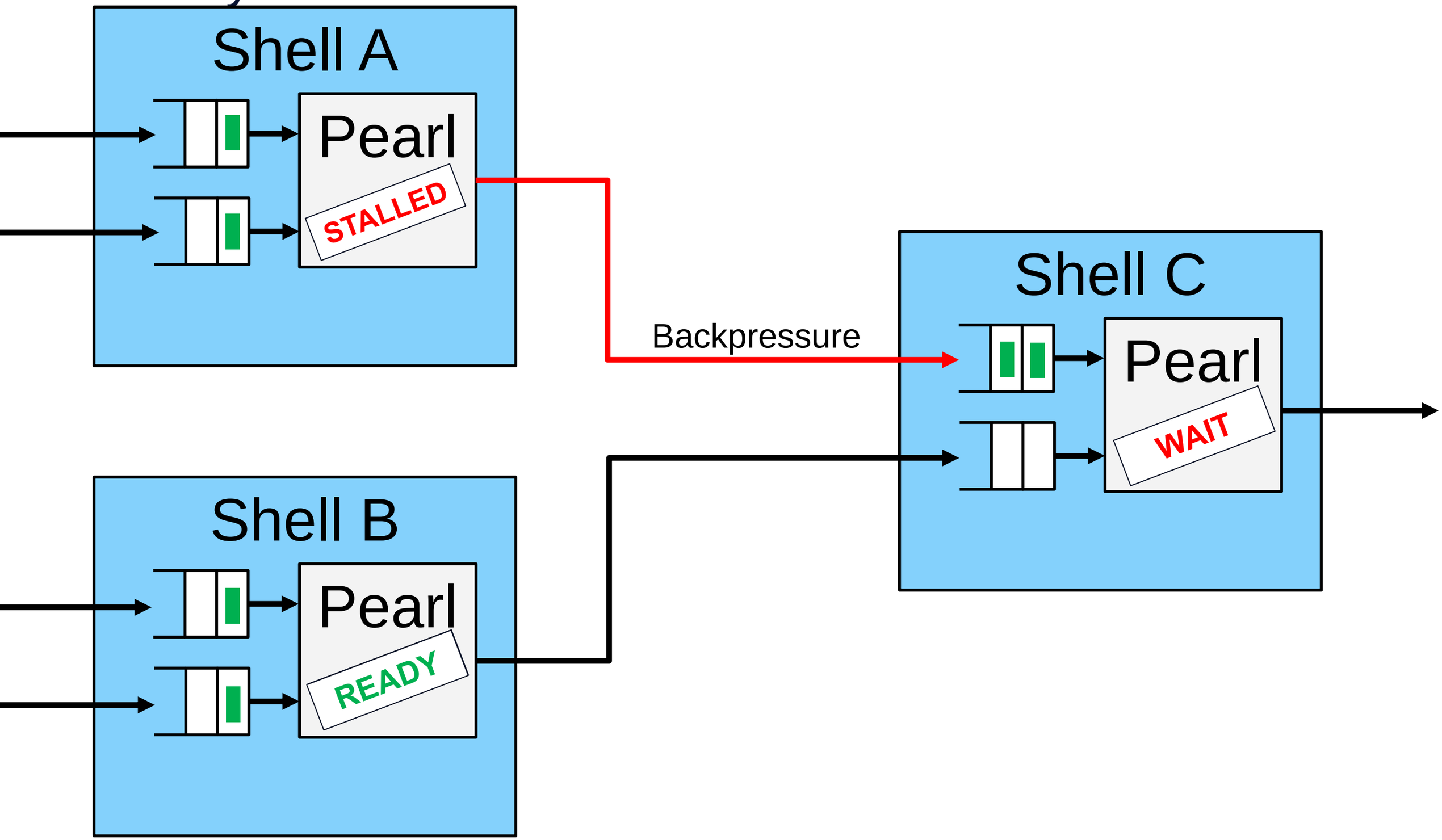
Latency Insensitive Communication Protocol



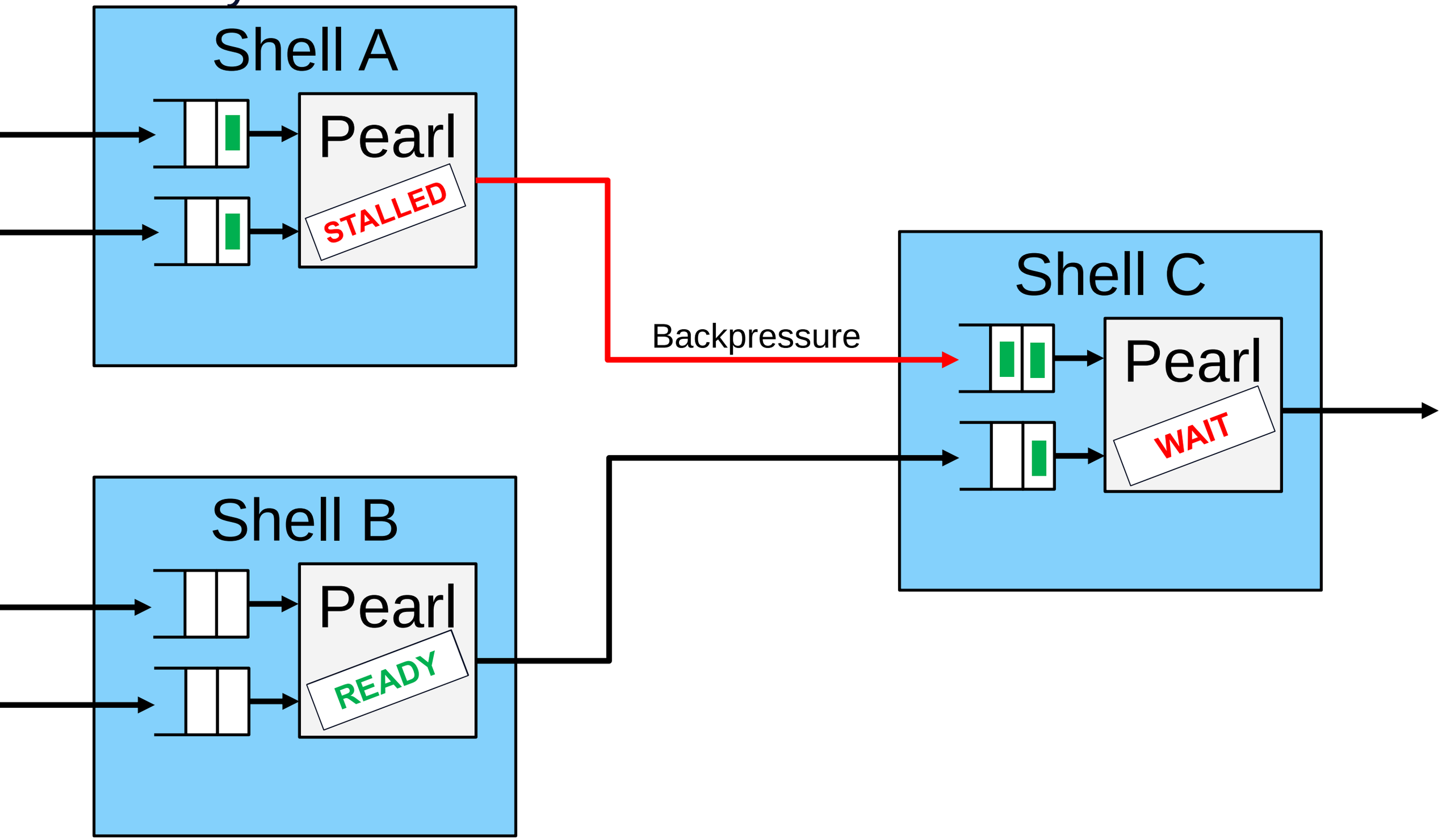
Latency Insensitive Communication Protocol



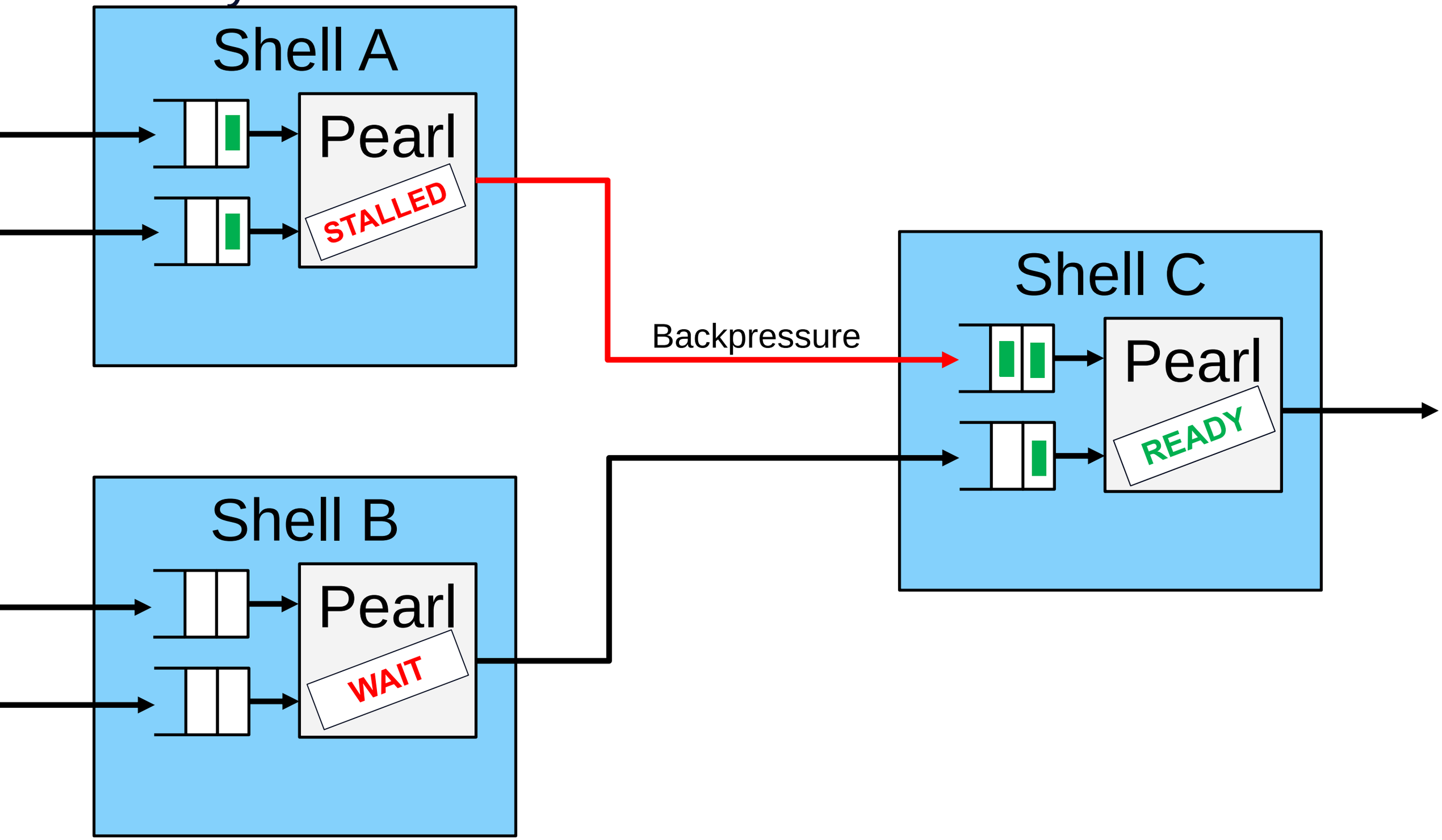
Latency Insensitive Communication Protocol



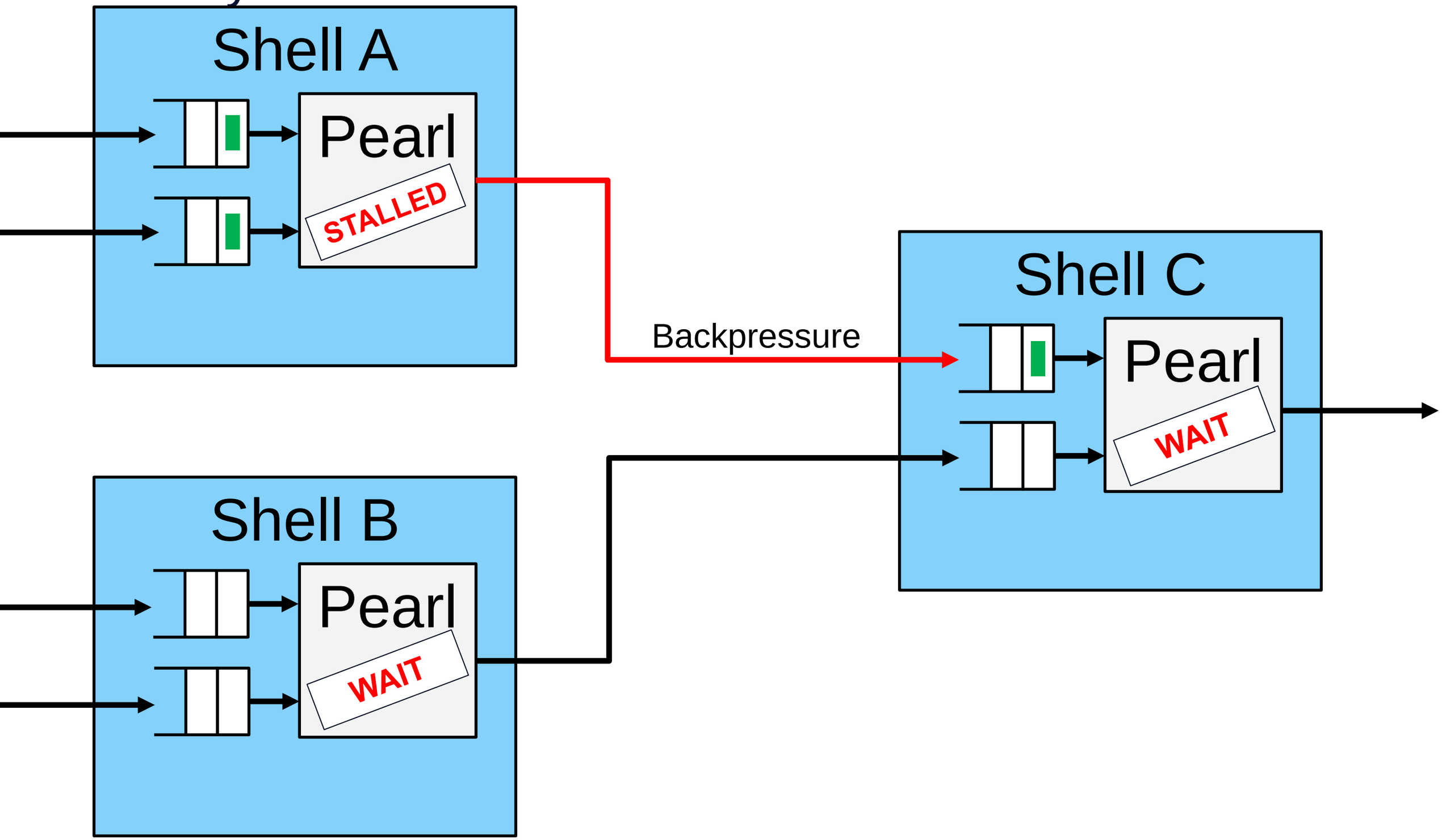
Latency Insensitive Communication Protocol



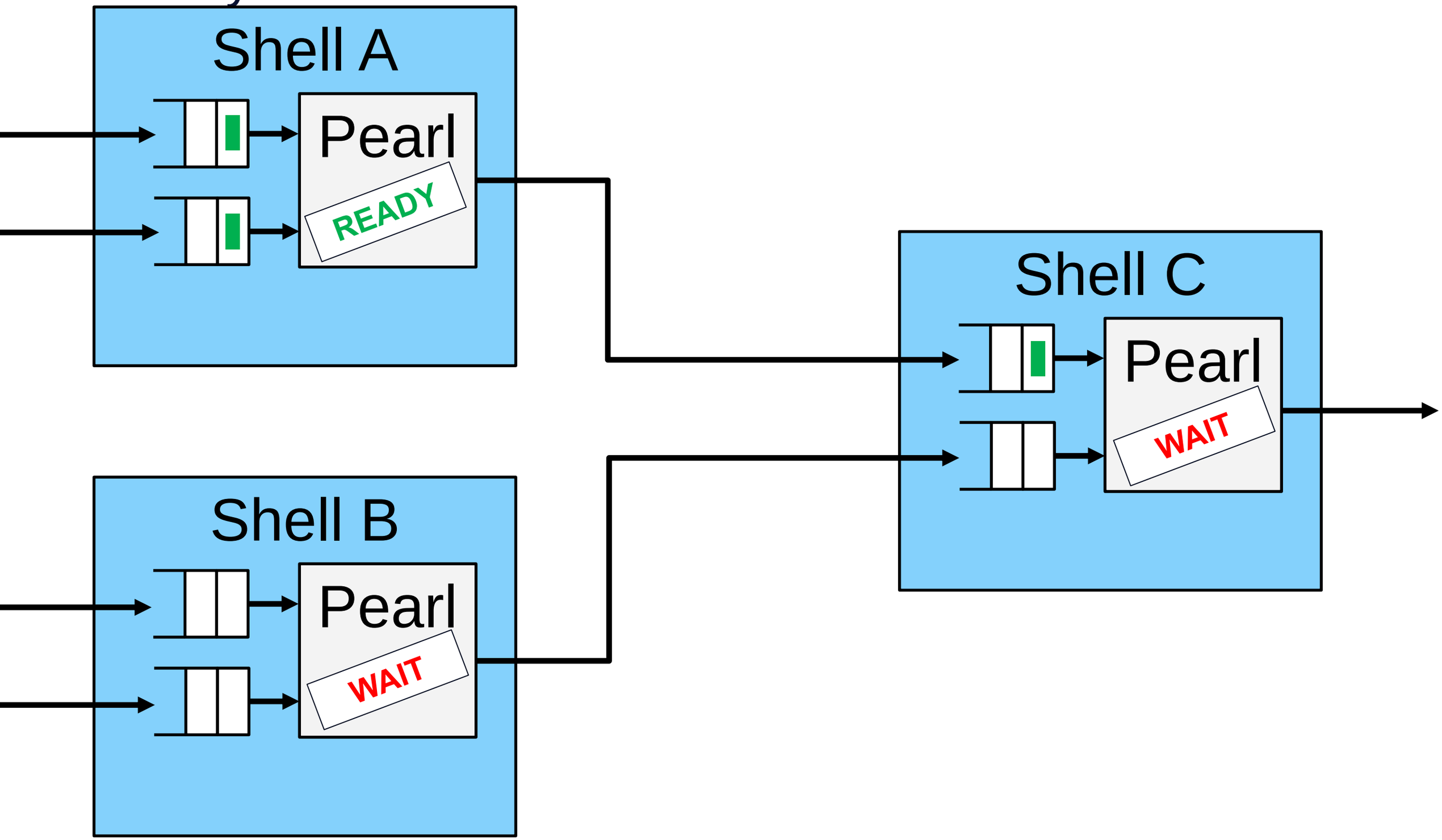
Latency Insensitive Communication Protocol



Latency Insensitive Communication Protocol



Latency Insensitive Communication Protocol

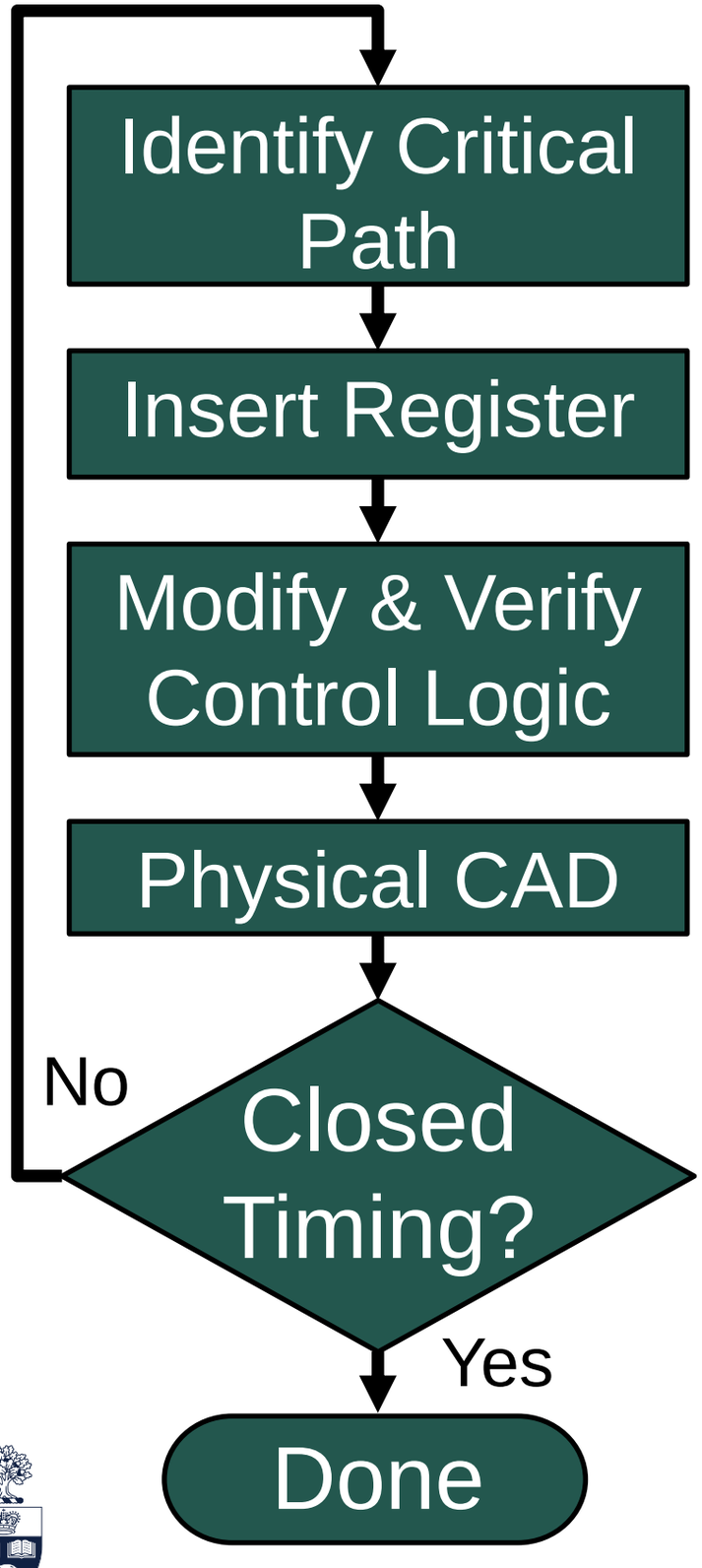


Latency Insensitive Design

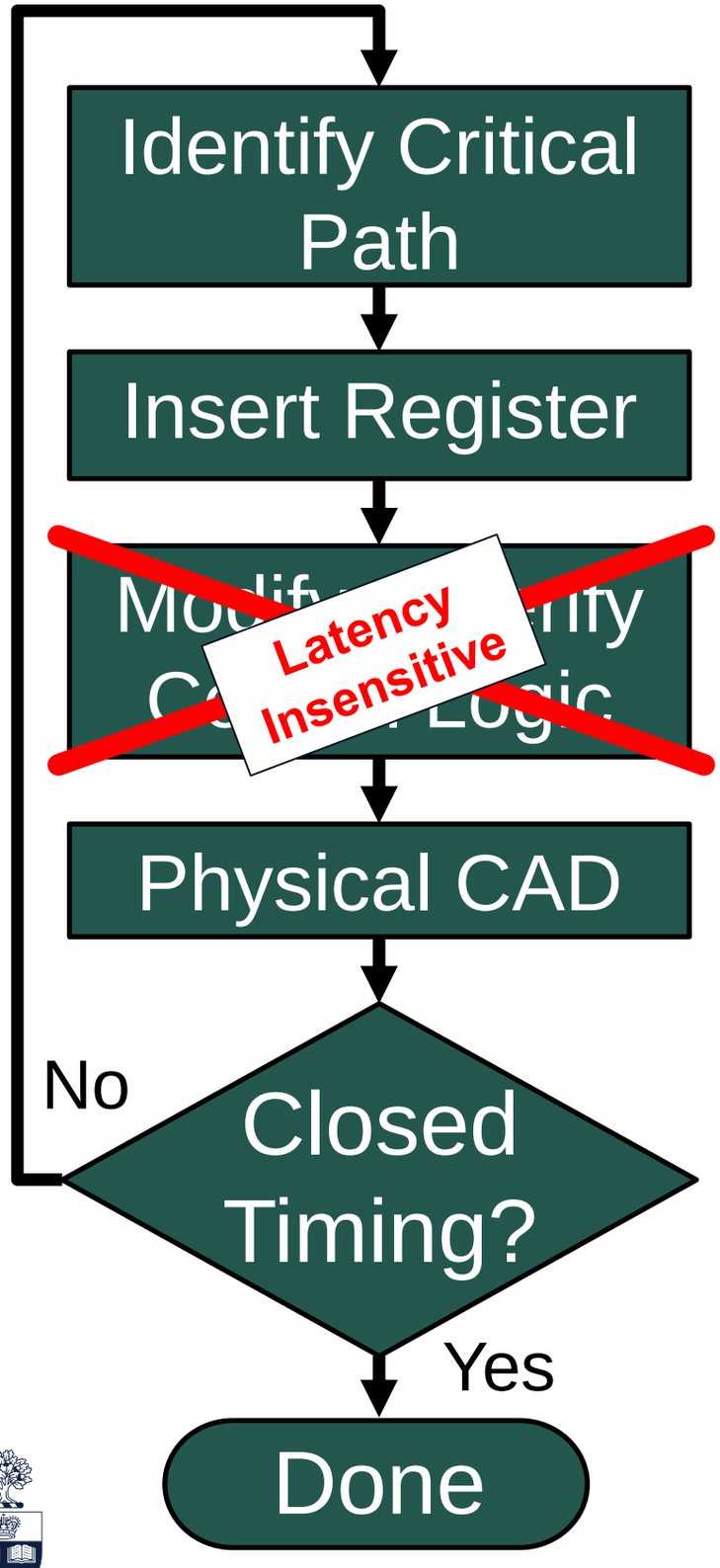
Advantages:

- Interconnect pipelining ***does not affect correctness***
  - Designers can still reason about system synchronously
  - Easy to pipeline late in the design flow
- Enhanced module re-use & composability
- Suitable for automation
  - Use existing CAD tools

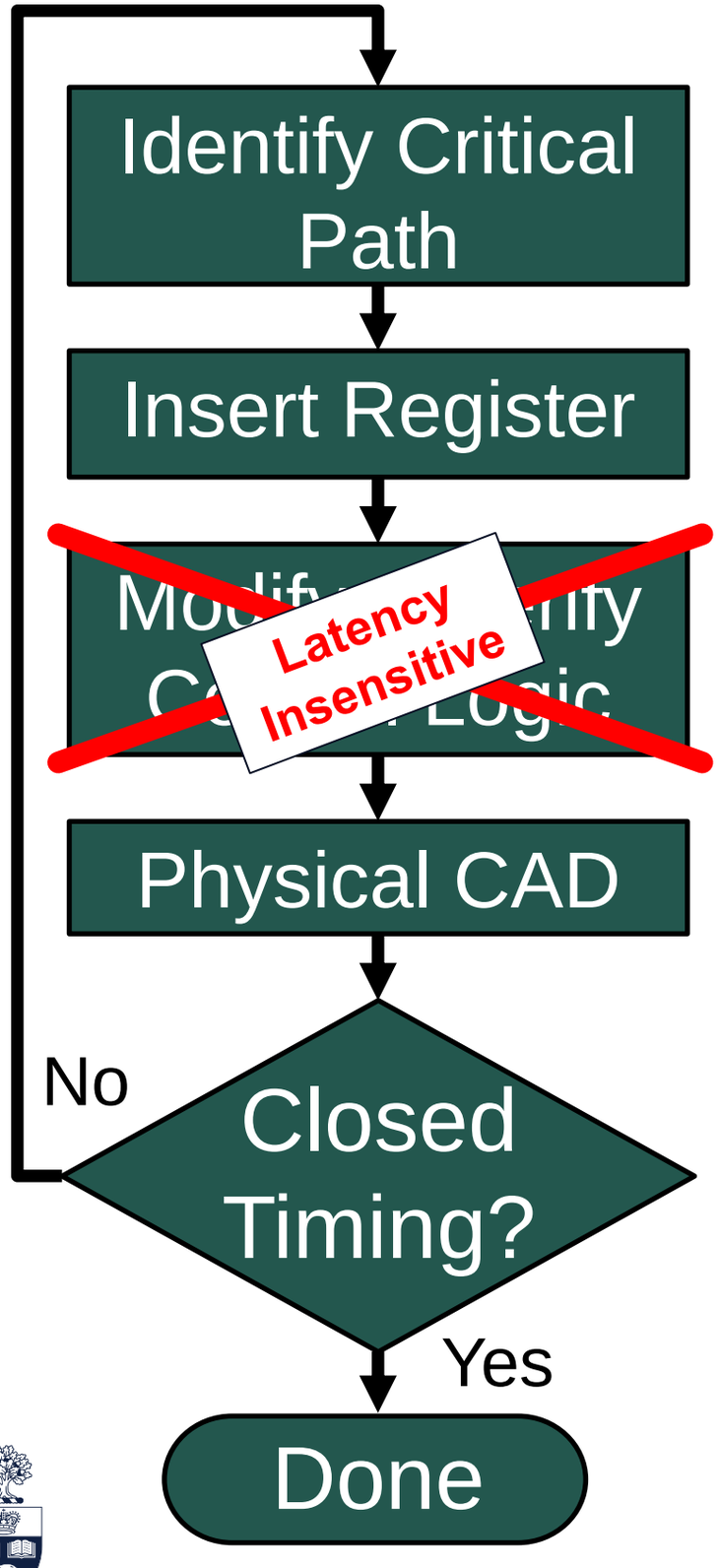
Interconnect Pipelining Automation

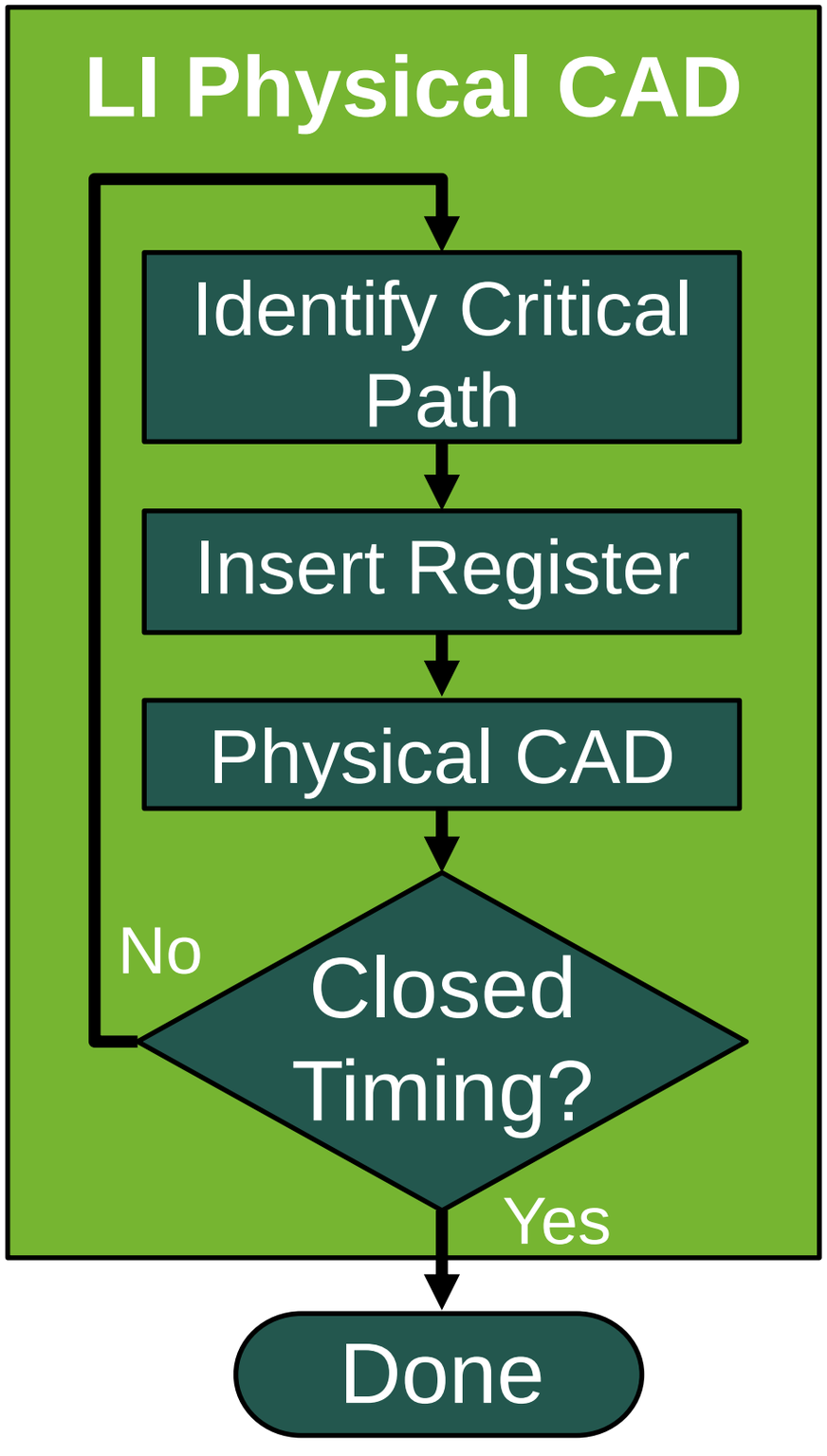


Interconnect Pipelining Automation



Interconnect Pipelining Automation





Latency Insensitive Design

Advantages:

- Interconnect pipelining ***does not affect correctness***
  - Designers can still reason about system synchronously
  - Easy to pipeline late in the design flow
- Enhanced module re-use & composability
- Suitable for automation
  - Use existing CAD tools
  - Fold interconnect pipelining into physical CAD Tools [Future work]

Latency Insensitive Design

Advantages:

- Interconnect pipelining ***does not affect correctness***
  - Designers can still reason about system synchronously
  - Easy to pipeline late in the design flow
- Enhanced module re-use & composability
- Suitable for automation
  - Use existing CAD tools
  - Fold interconnect pipelining into physical CAD Tools [Future work]

Disadvantages:

- Area/Speed overhead versus hand-tuned design
- Must verify sufficient throughput

Latency Insensitive Design

Trade off:

- Implementation efficiency for designer productivity

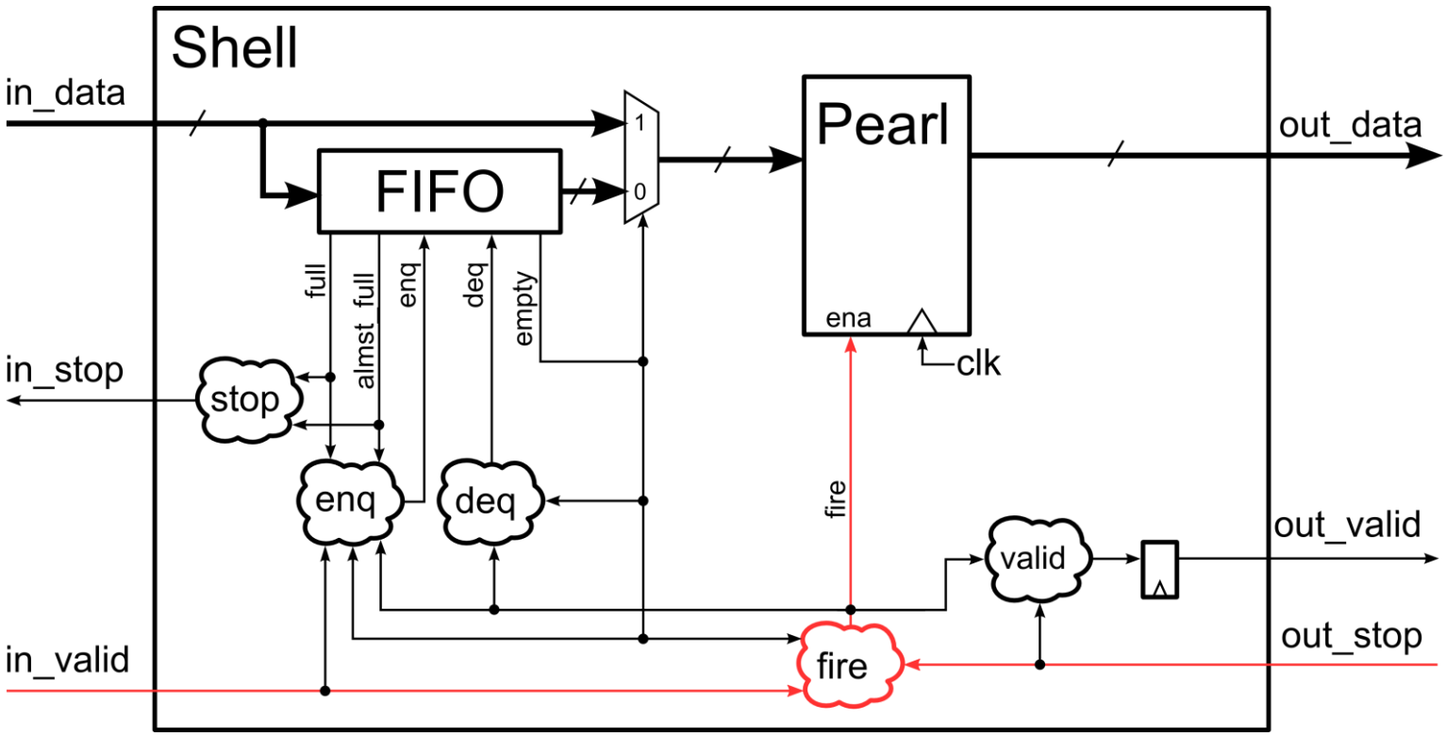
Key question of this work:

- What are the overheads of LI design on FPGAs?

Latency Insensitive Implementation

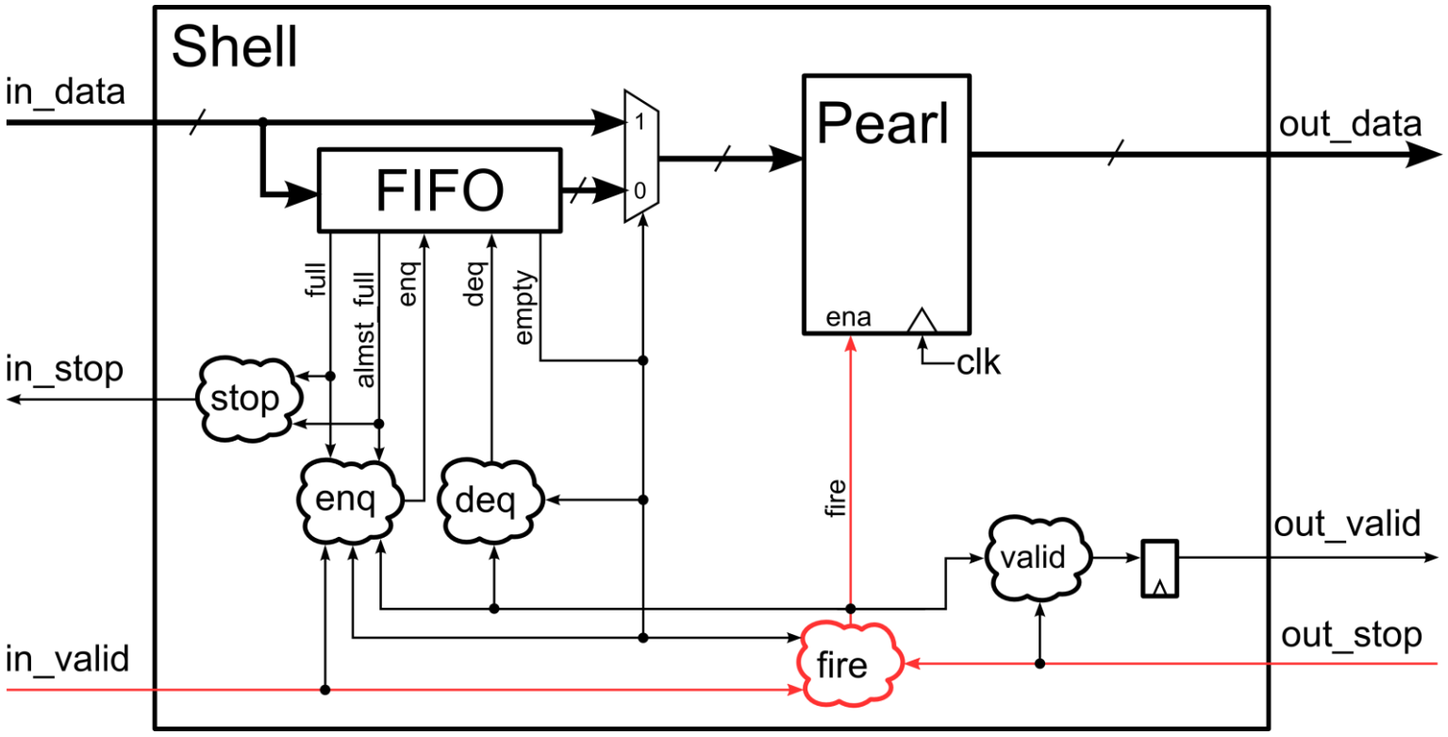
Baseline Shell Implementation

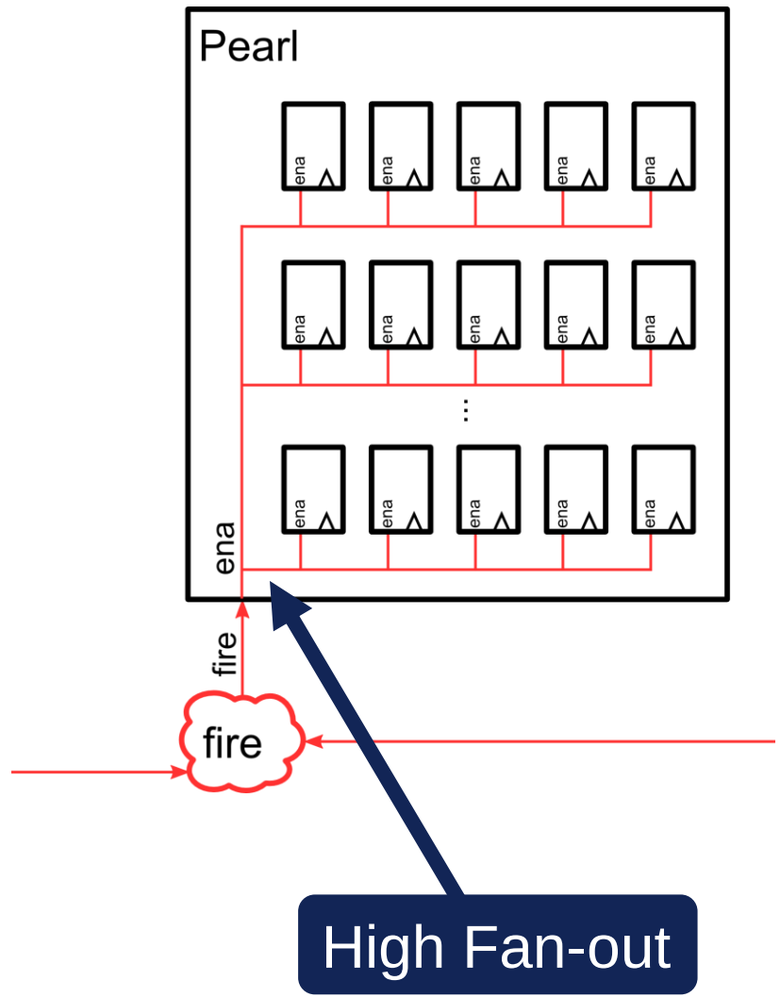
- ASIC LI design stalls modules by clock gating
  - Use ‘Clock Enable’ on FPGAs



Baseline Shell Implementation

- ASIC LI design stalls modules by clock gating
  - Use ‘Clock Enable’ on FPGAs

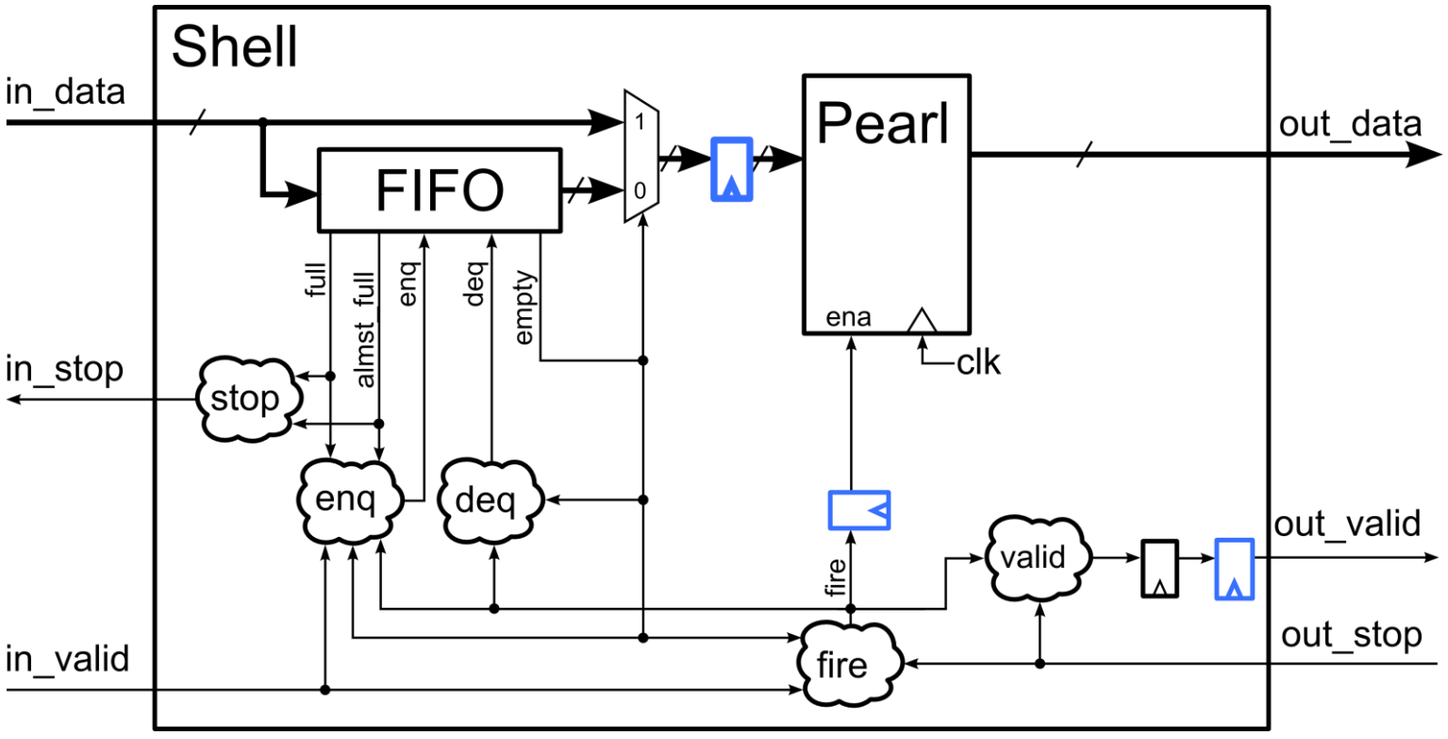


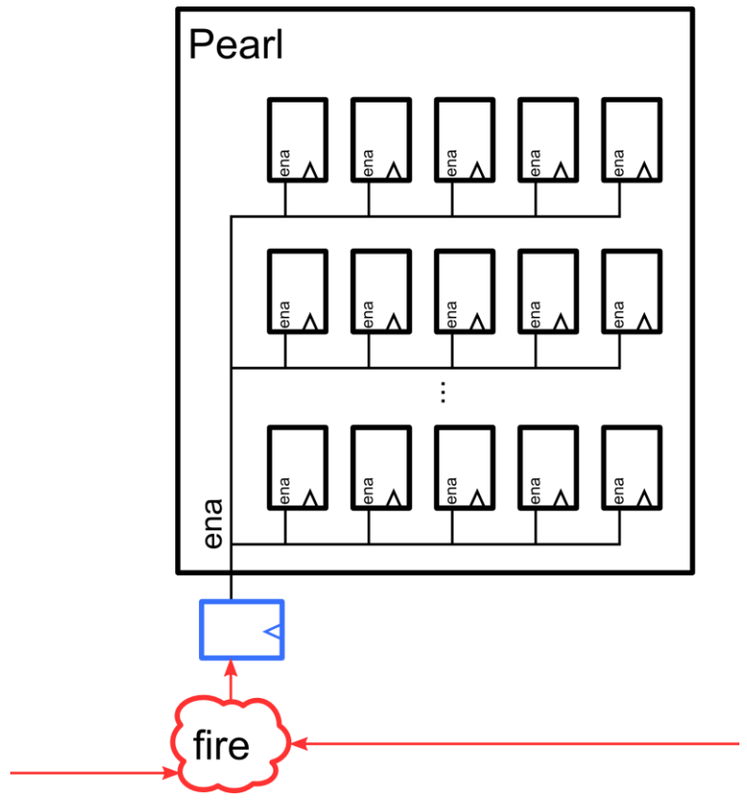


- ‘Clock Enable’ becomes timing critical
  - Fans out to all registers in pearl
  - Connected to upstream and downstream modules

Optimized Shell Implementation

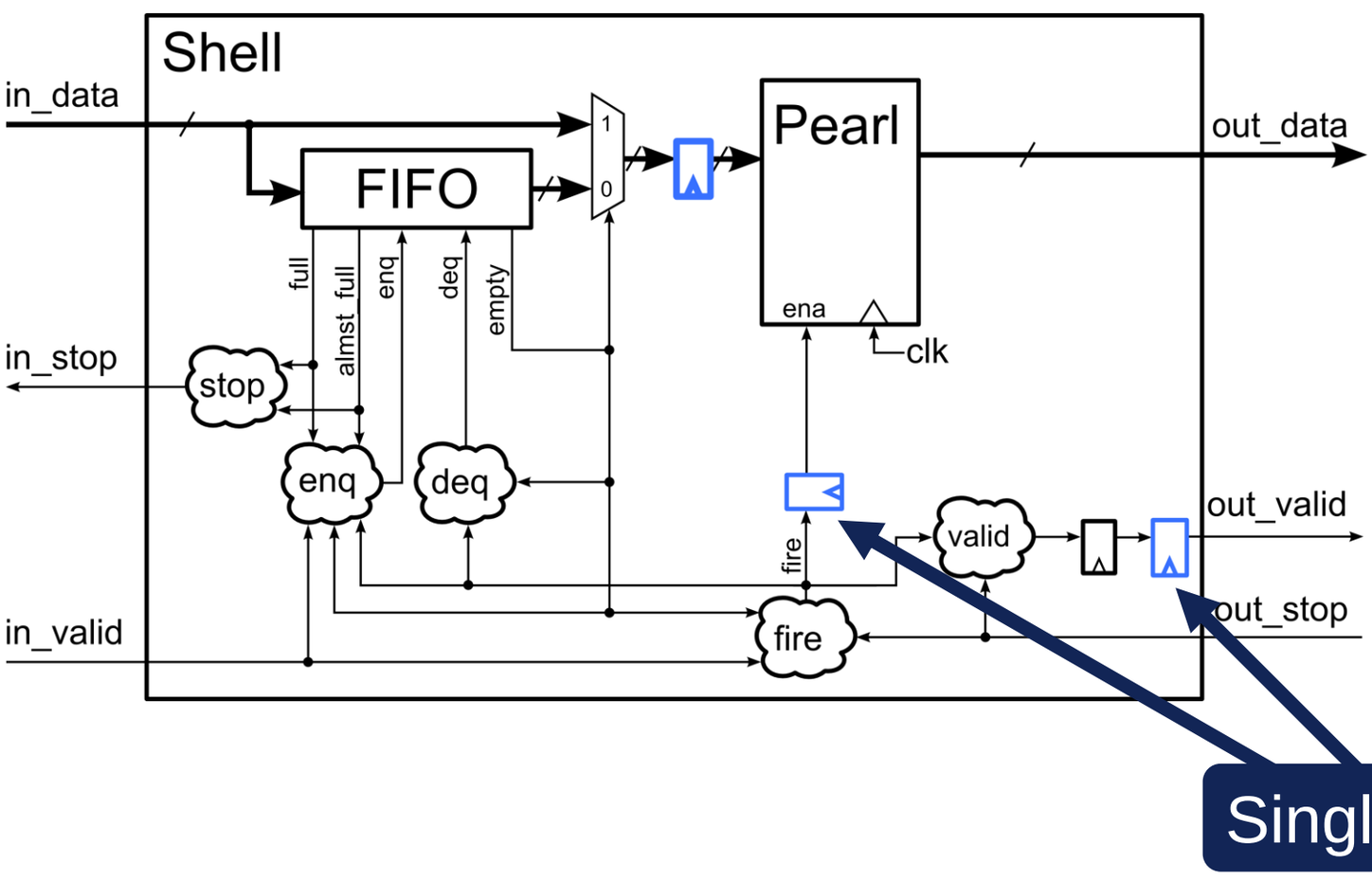
- Break timing path before it becomes high fan-out
  - Insert additional registers in Shell

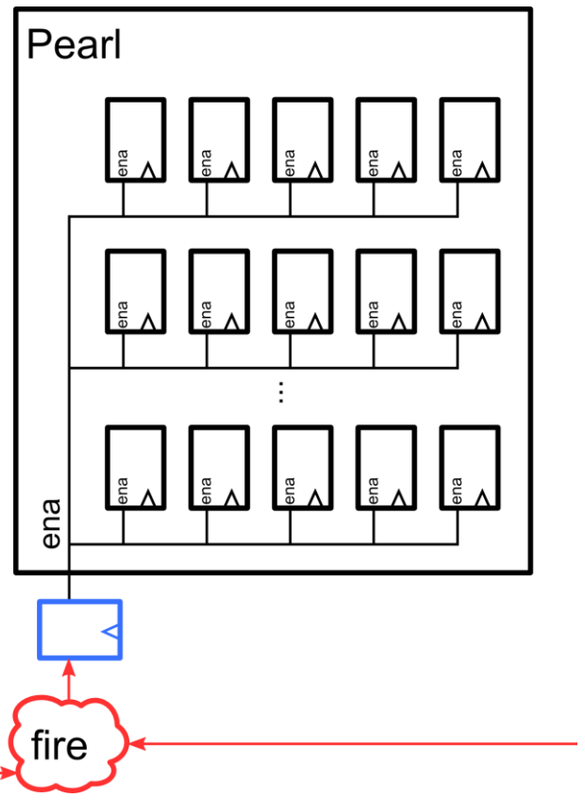




Optimized Shell Implementation

- Break timing path before it becomes high fan-out
  - Insert additional registers in Shell

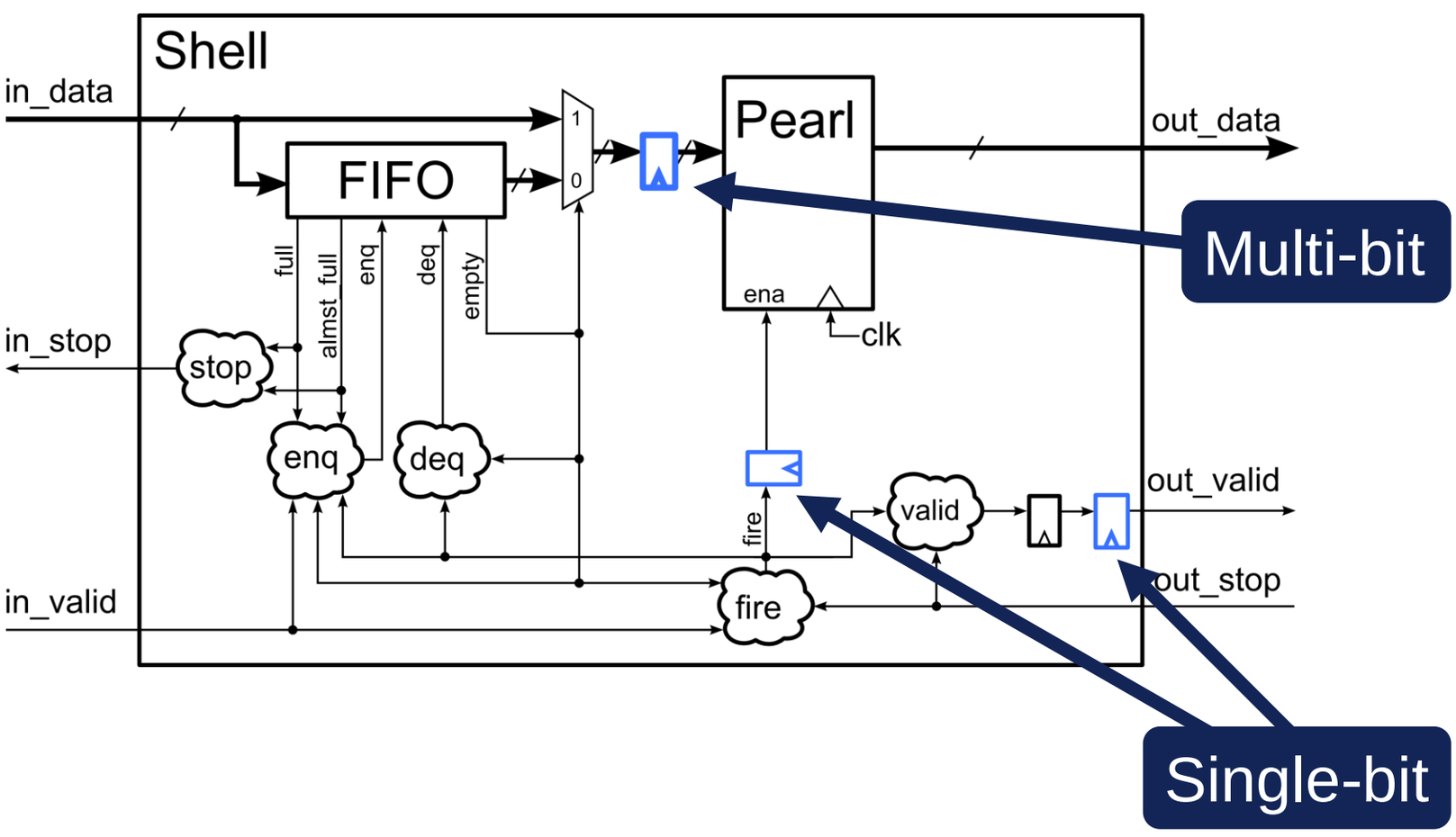


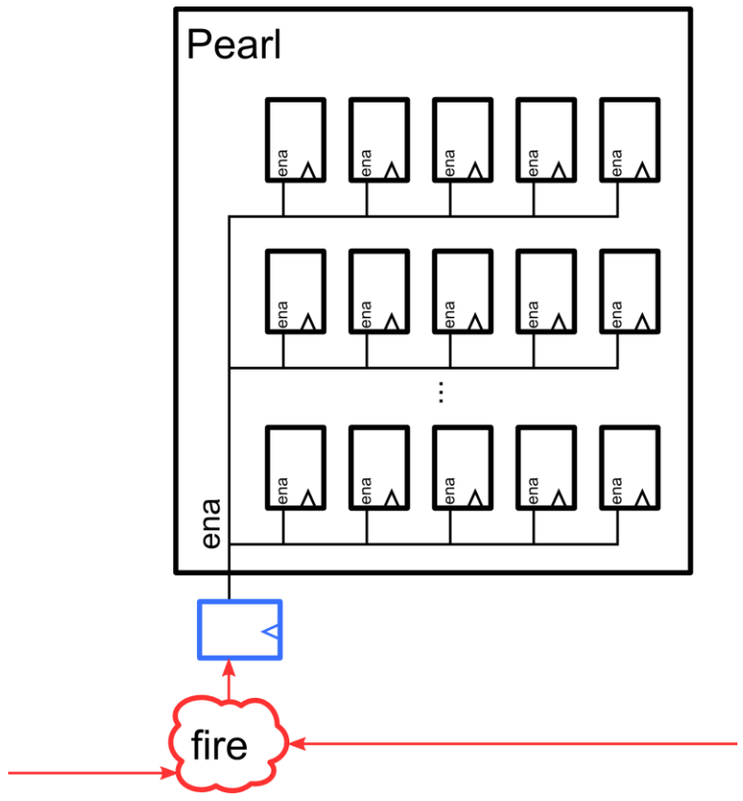


Single-bit

Optimized Shell Implementation

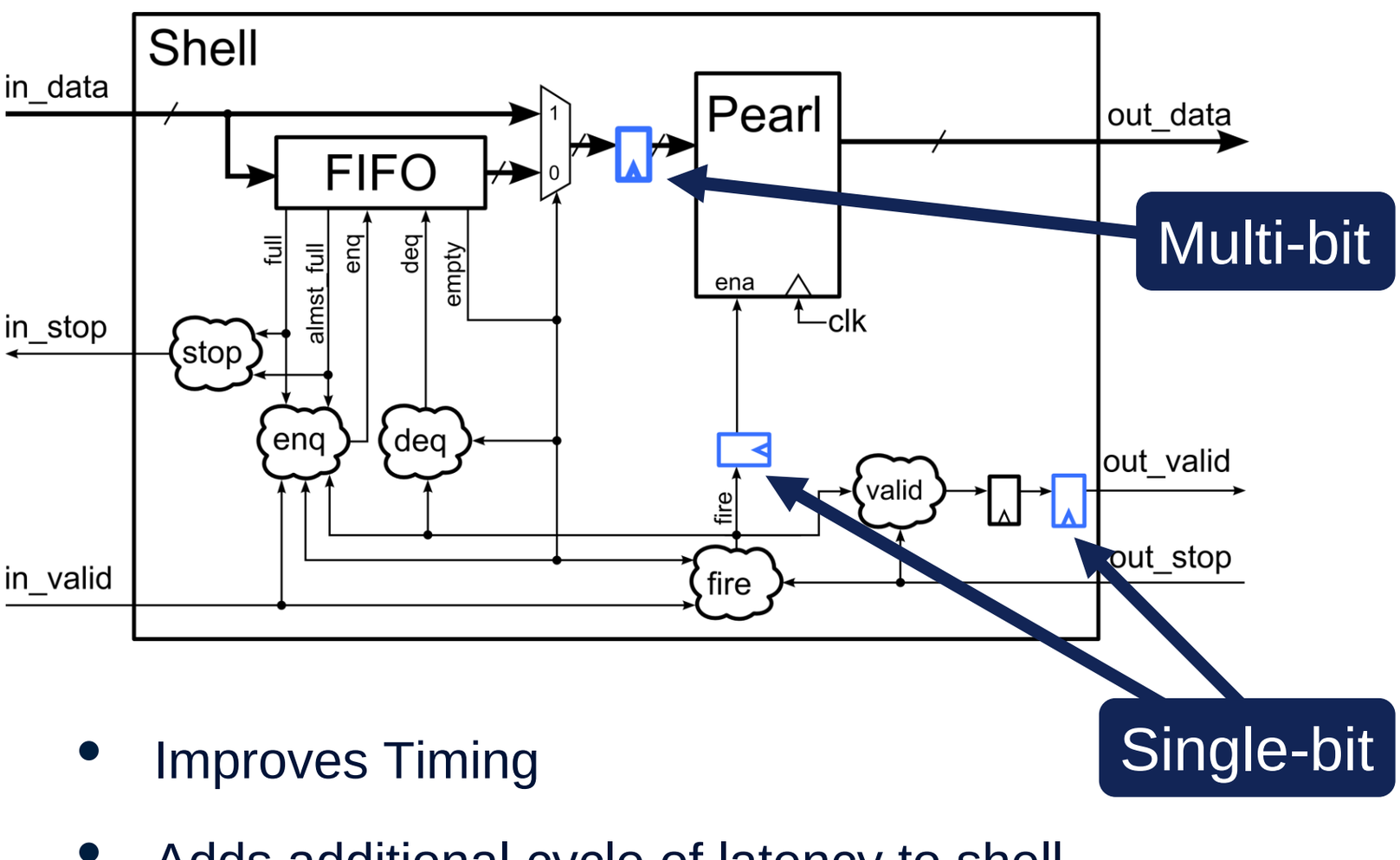
- Break timing path before it becomes high fan-out
  - Insert additional registers in Shell

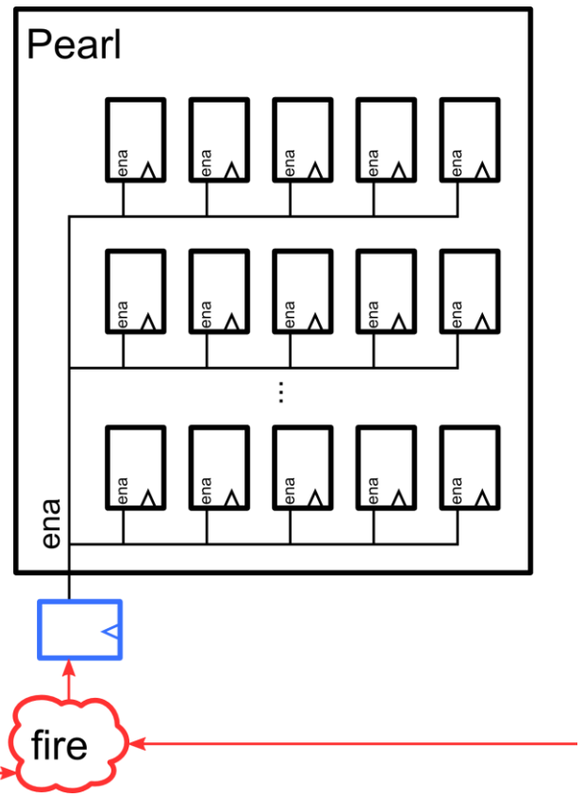




Optimized Shell Implementation

- Break timing path before it becomes high fan-out
  - Insert additional registers in Shell

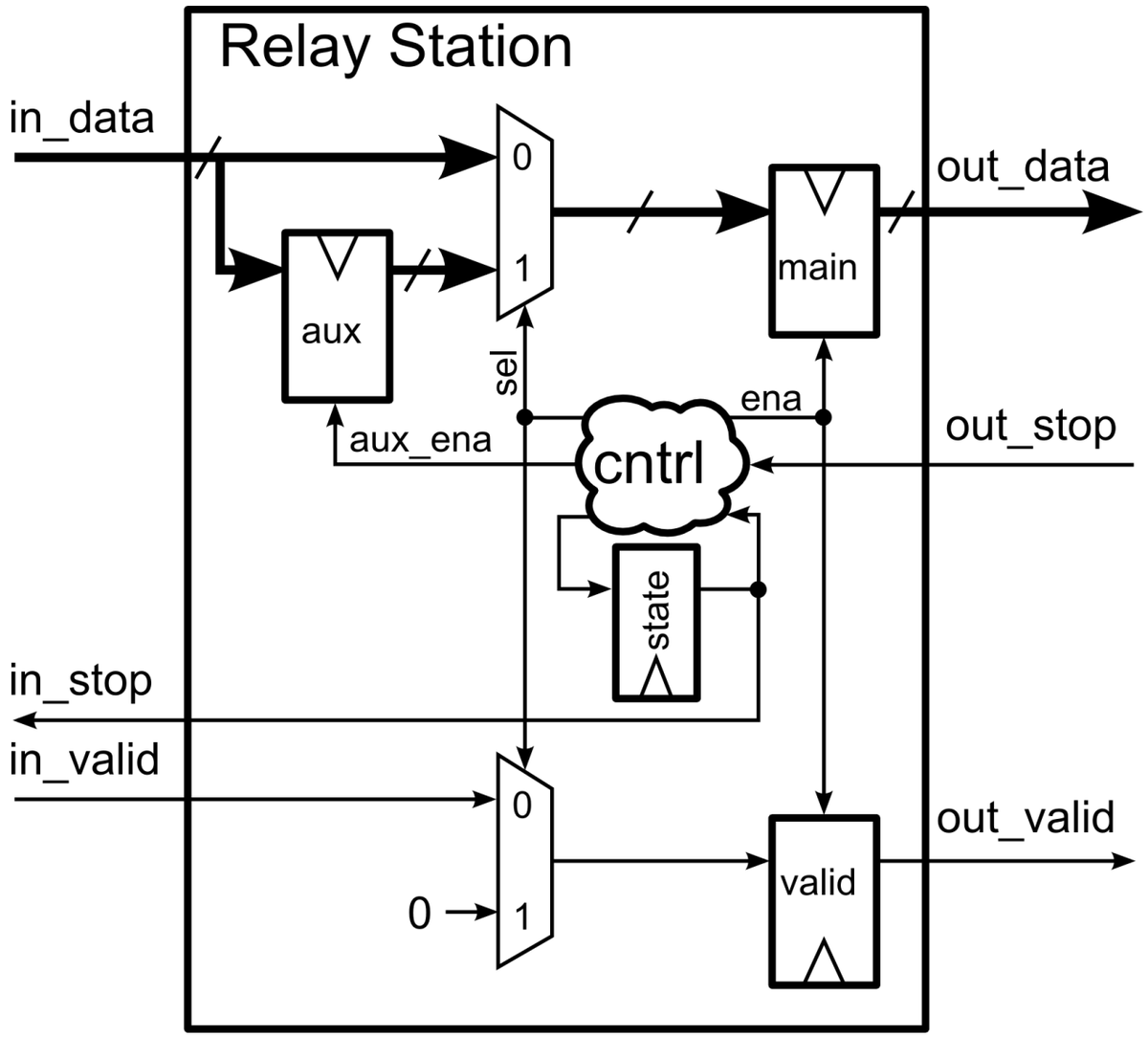




- Improves Timing
- Adds additional cycle of latency to shell

Relay Station (RS) Implementation

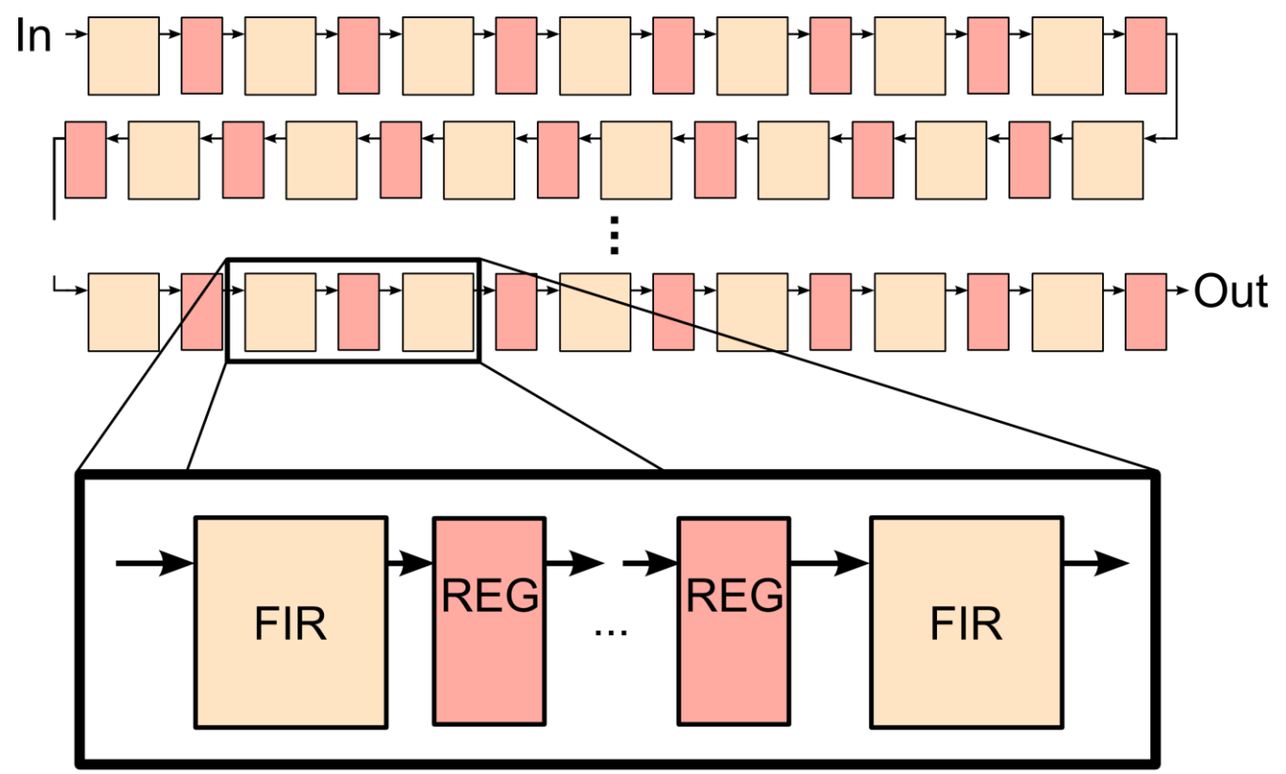
- Analogous to conventional pipeline register
- Additional logic to:
  - Handle ‘valid’ and ‘stop’ bits
  - Store in-flight data when facing backpressure (avoids stalling)



FIR Design Example

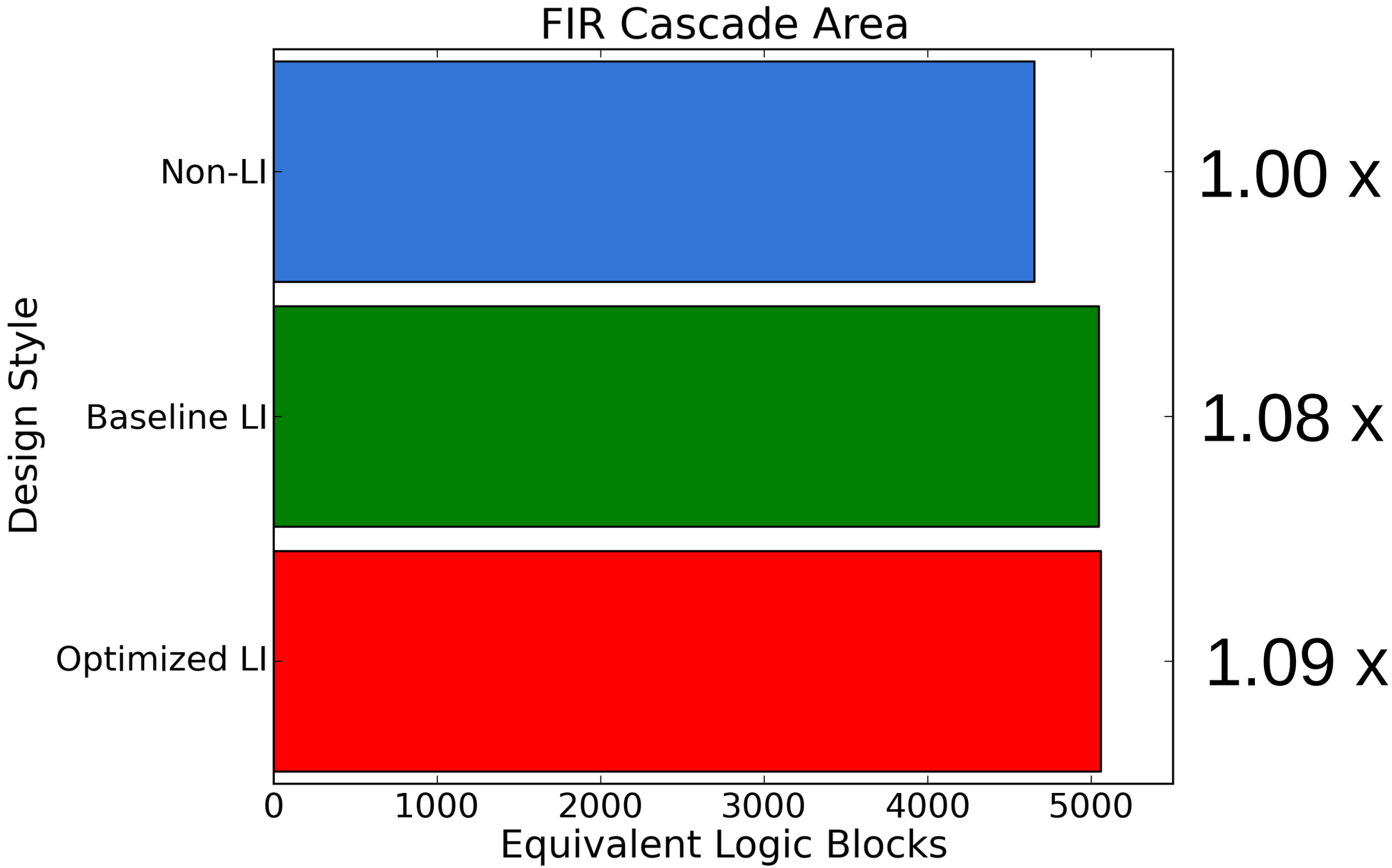
Cascaded FIR Case Study

- Pearl: FIR filter
- Design: 49 cascaded FIR filters
- Used as a high speed design example
  - Investigate the frequency impact of LI design
  - Allow comparison of LI and non-LI pipelining

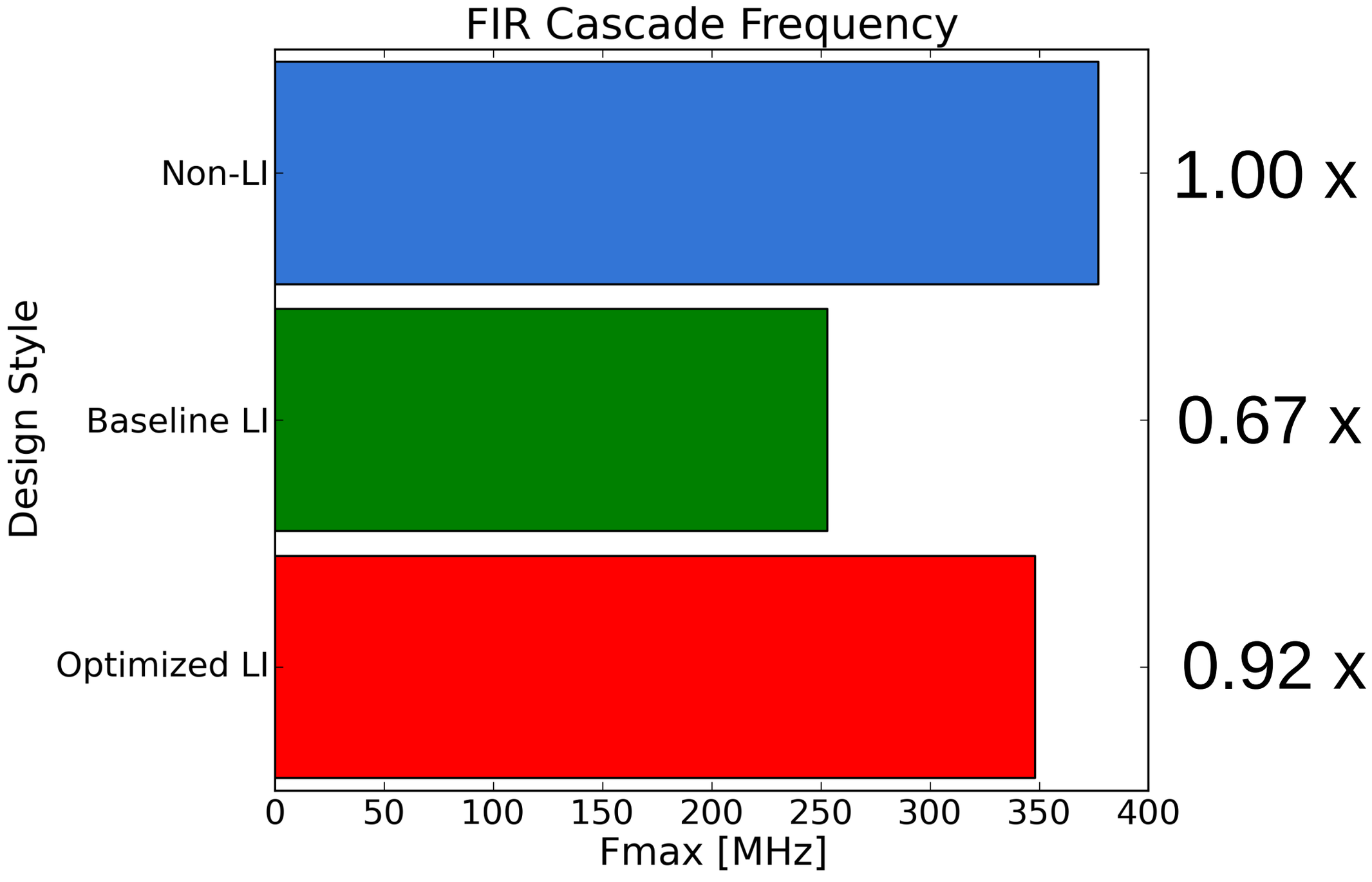


Resources	EP4sGX230 Utilization
Logic Blocks	51%
DSP Blocks	99%
M9K Blocks	<1%
M144K Blocks	0%

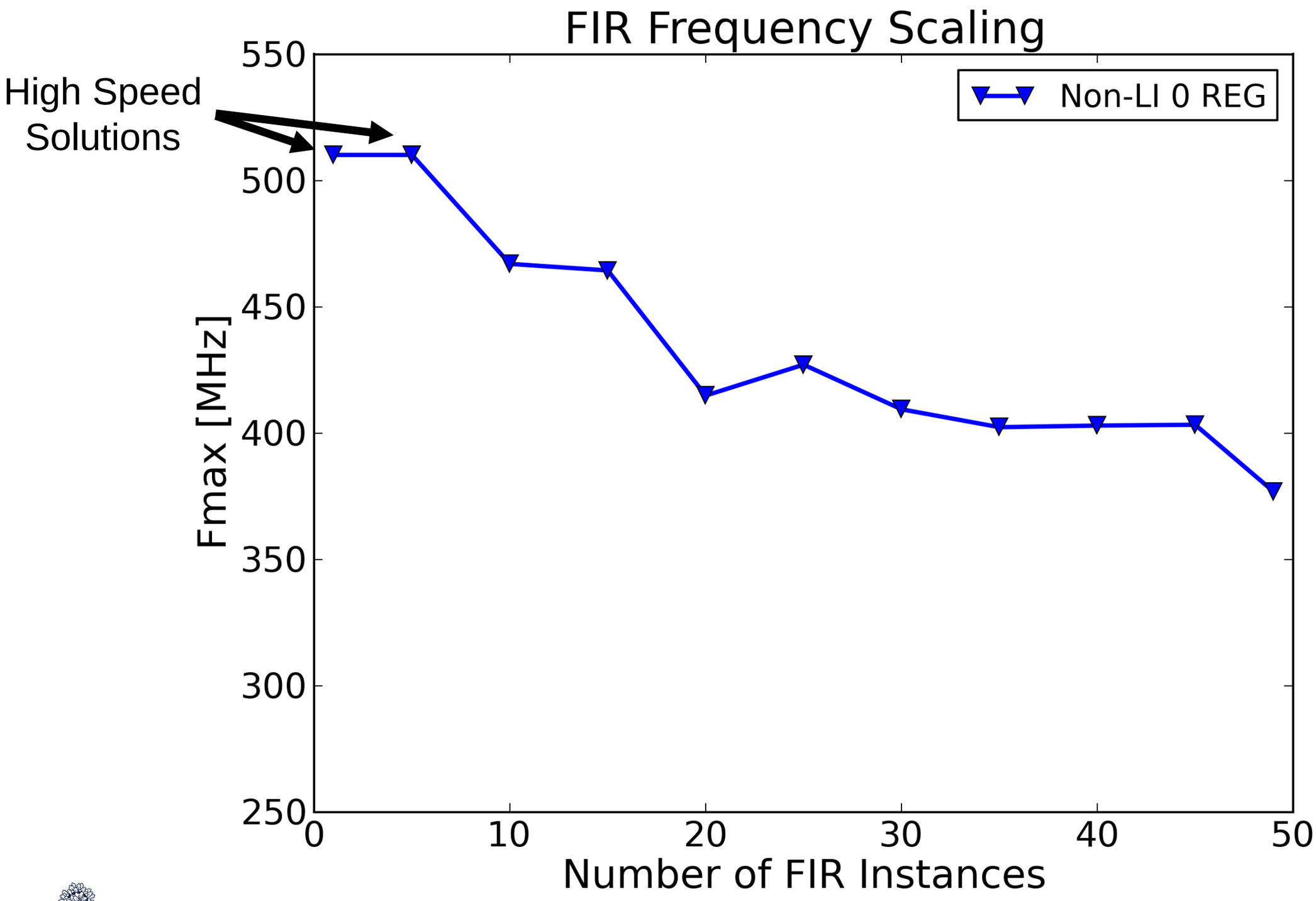
Cascaded FIR Case Study - Area



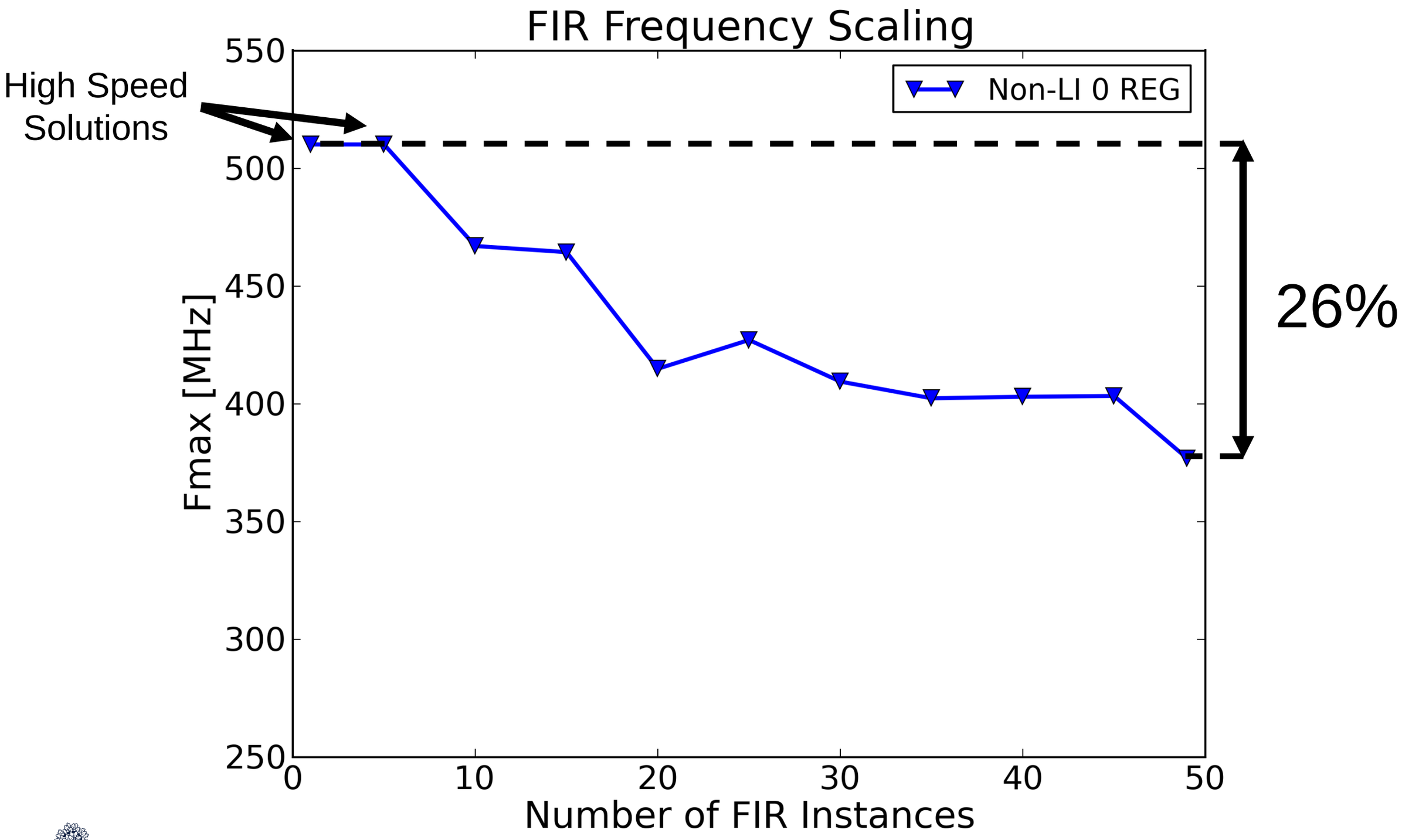
Cascaded FIR Case Study - Frequency



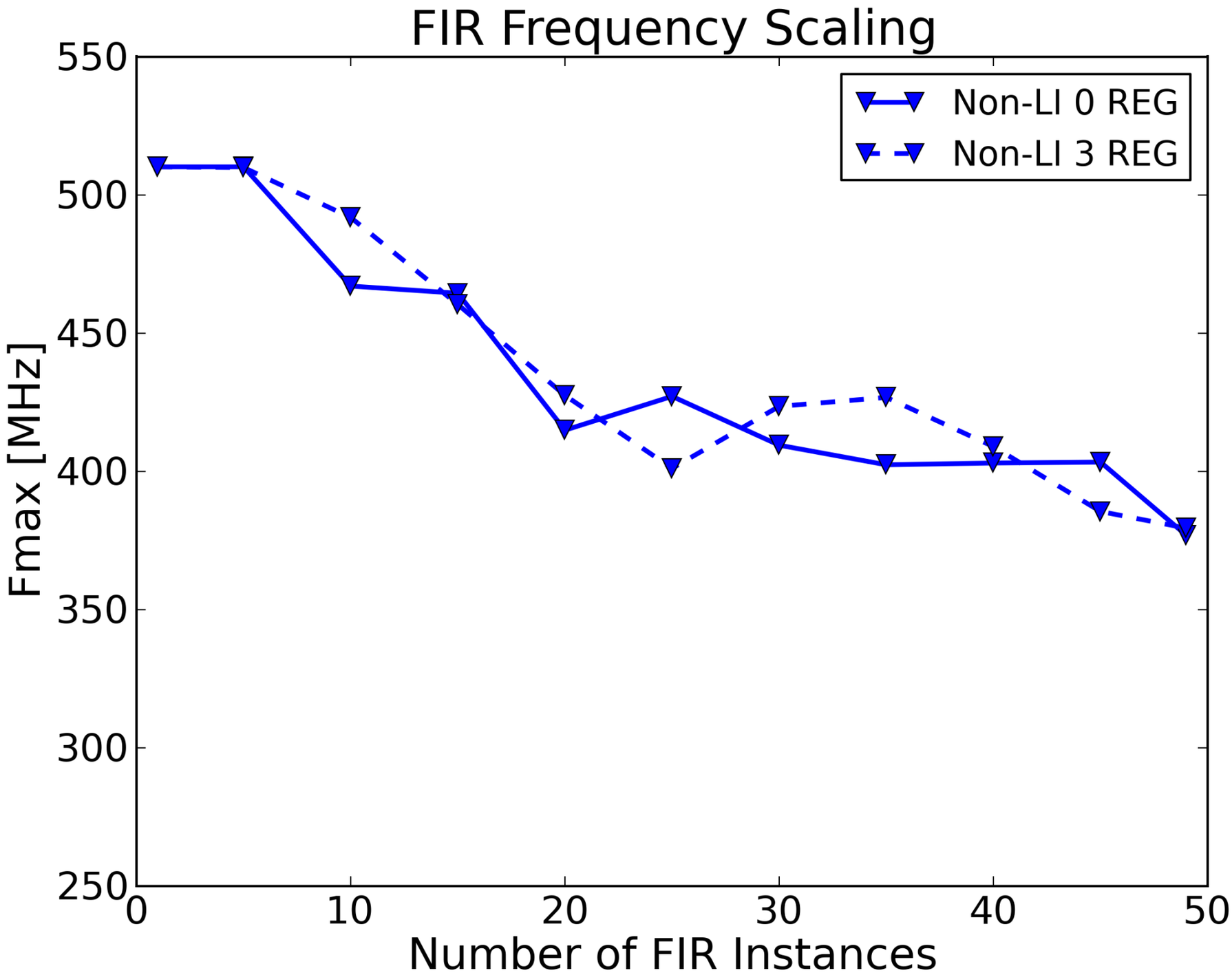
Cascaded FIR Case Study - Frequency



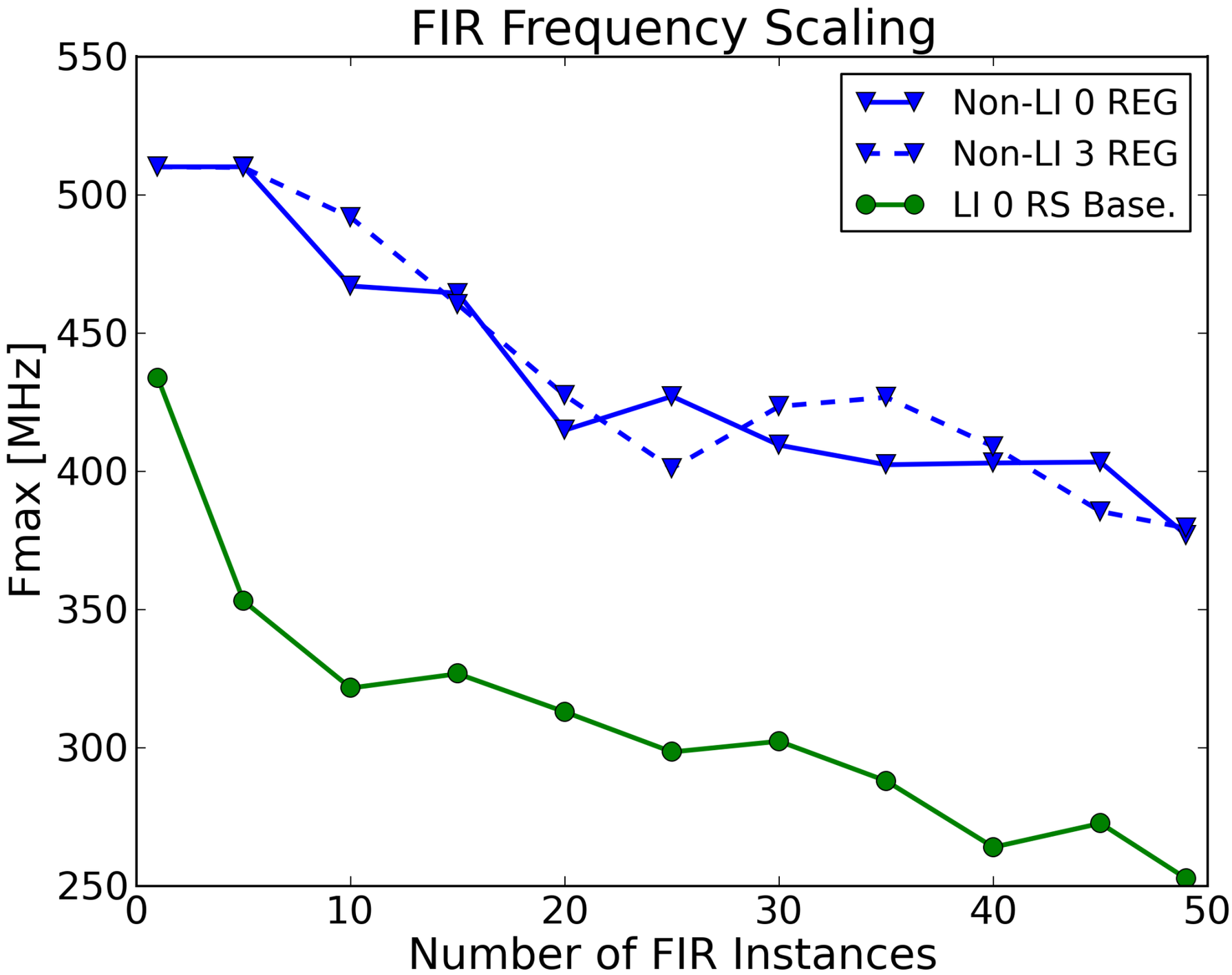
Cascaded FIR Case Study - Frequency



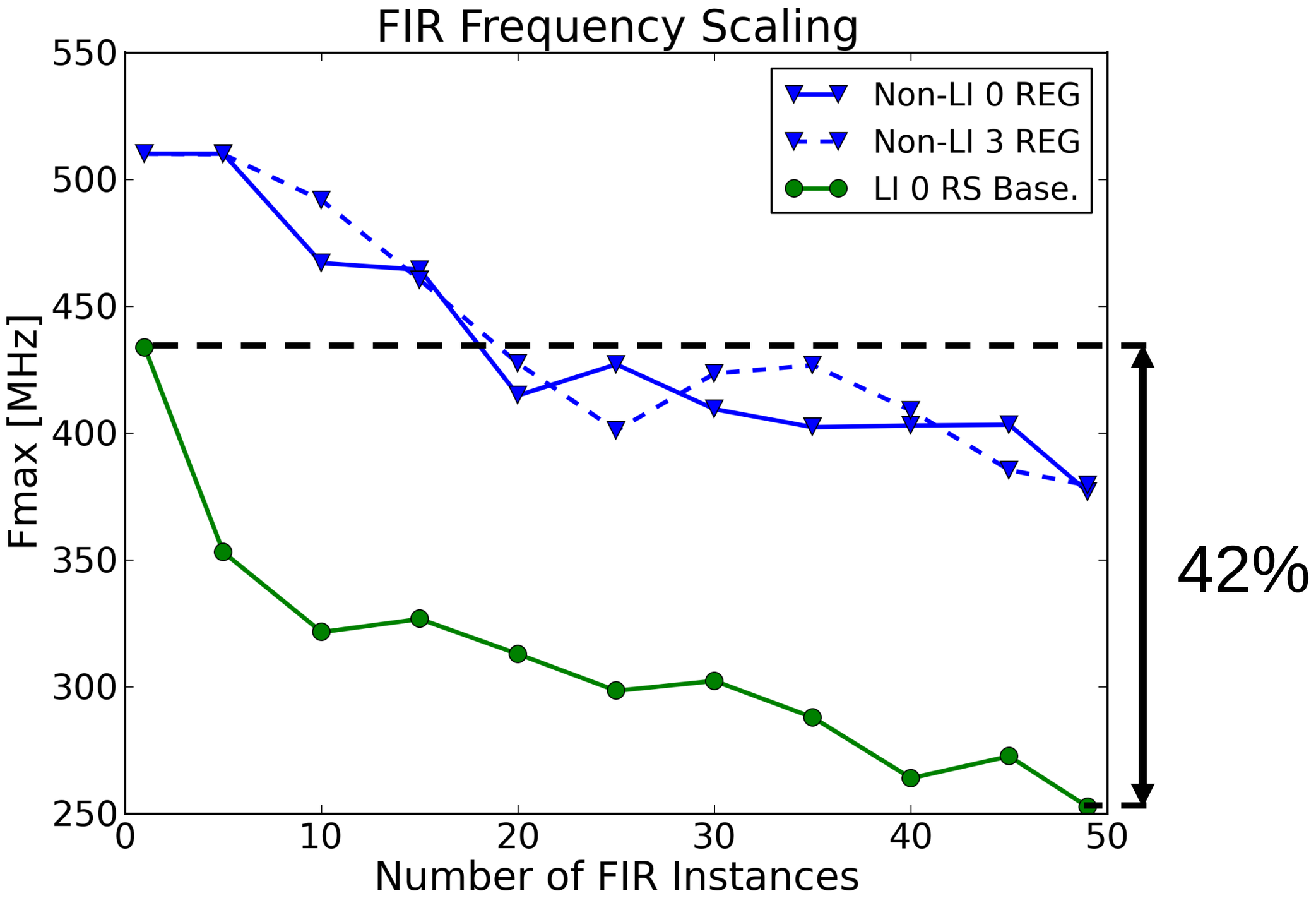
Cascaded FIR Case Study - Frequency



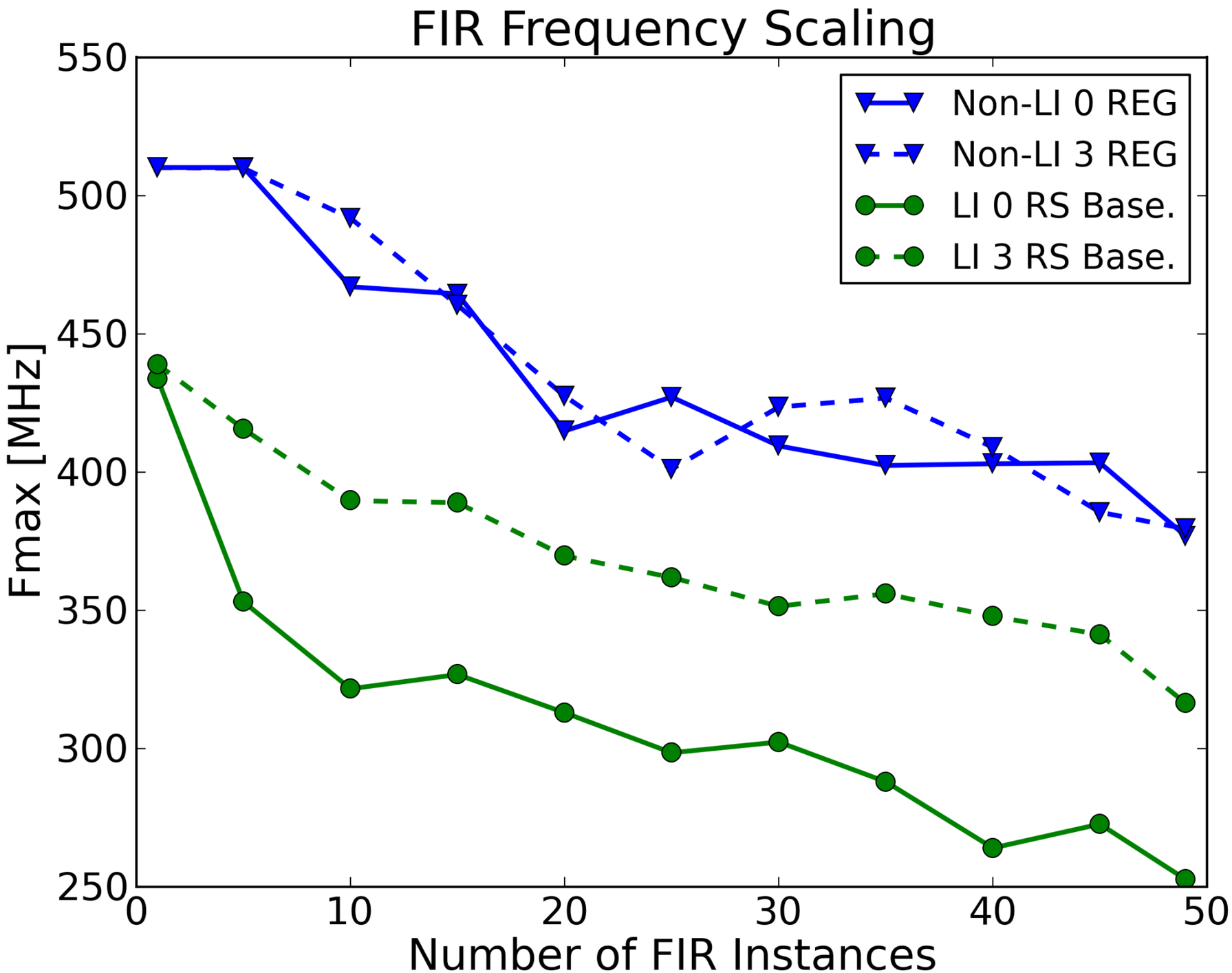
Cascaded FIR Case Study - Frequency



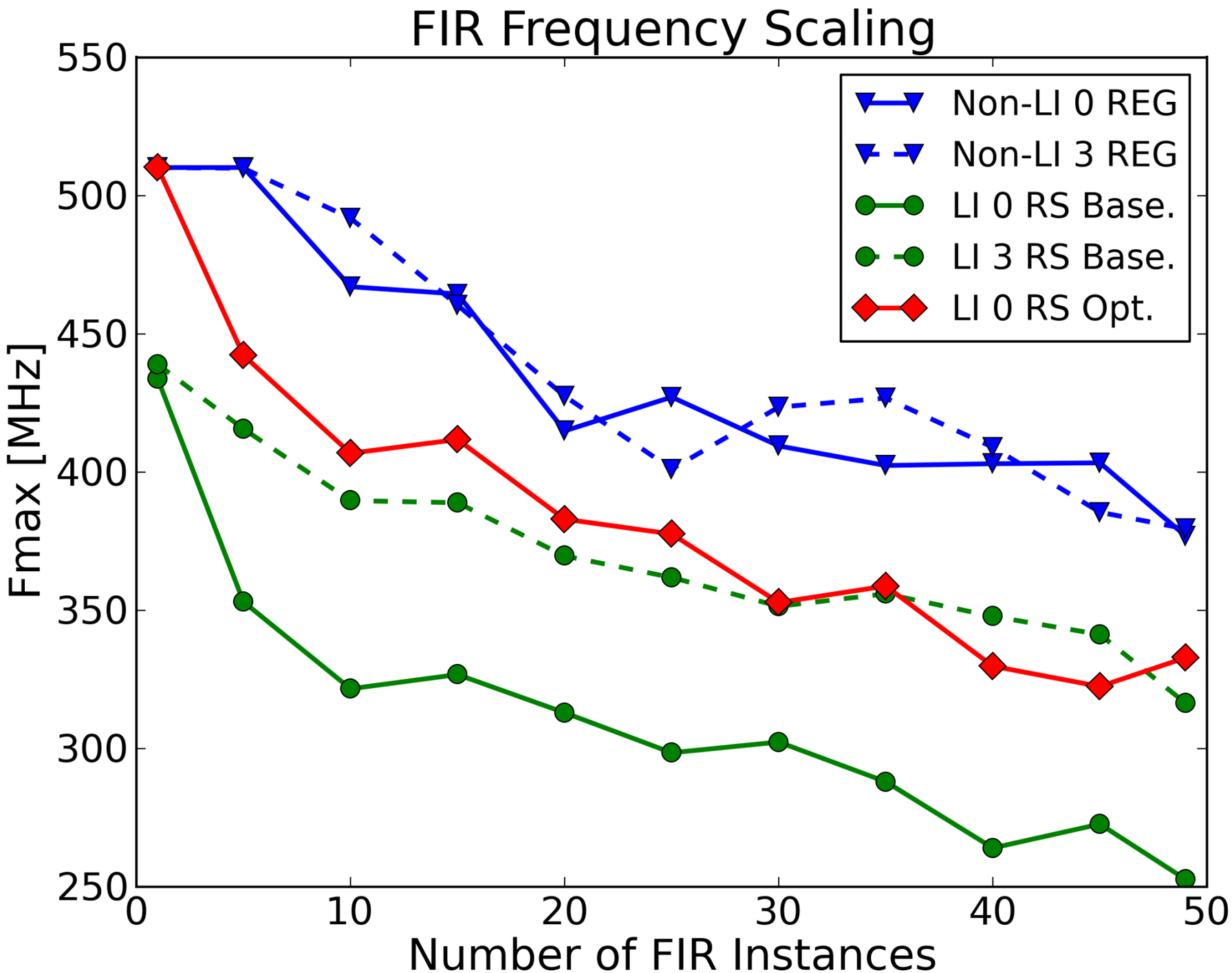
Cascaded FIR Case Study - Frequency



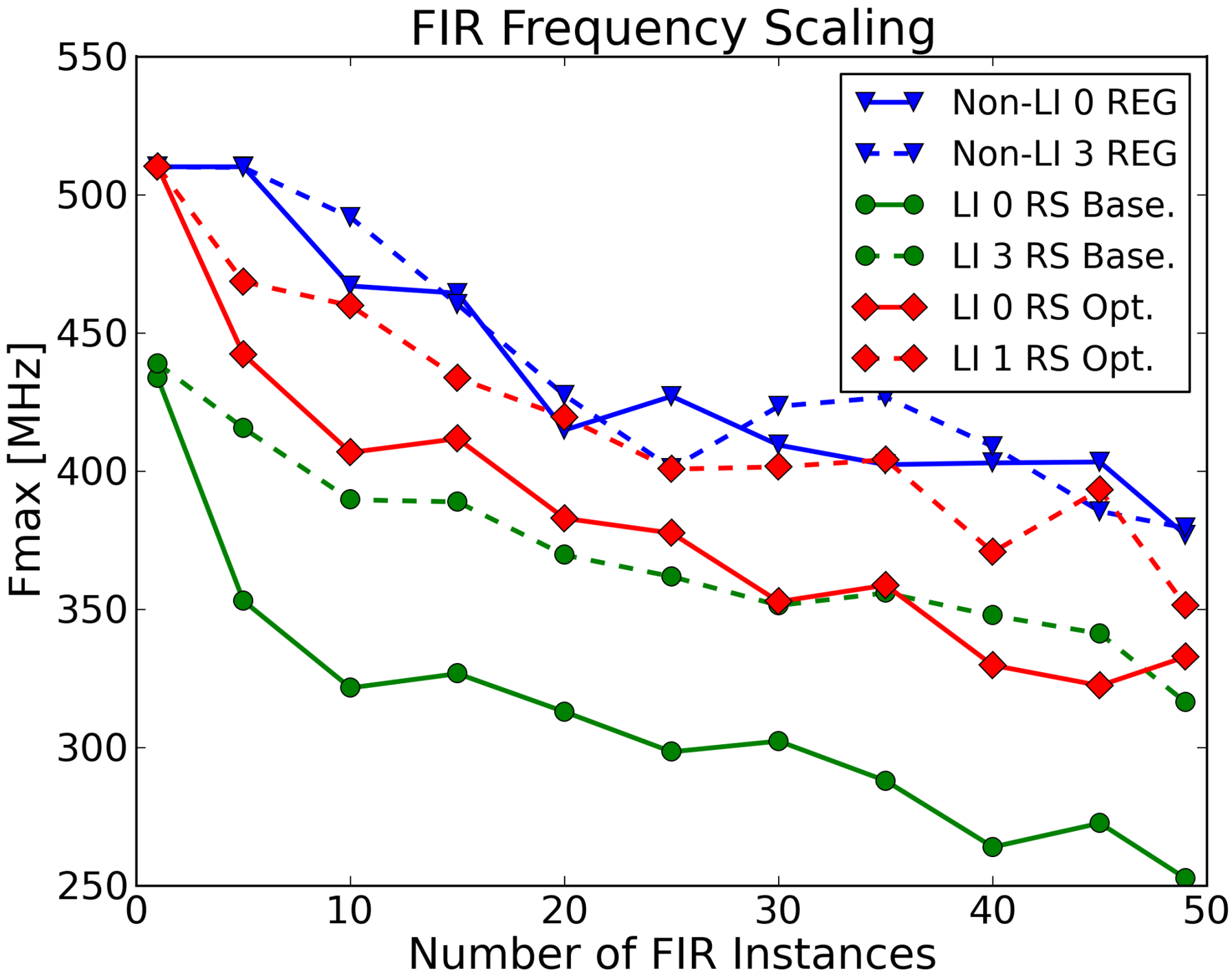
Cascaded FIR Case Study - Frequency



Cascaded FIR Case Study - Frequency

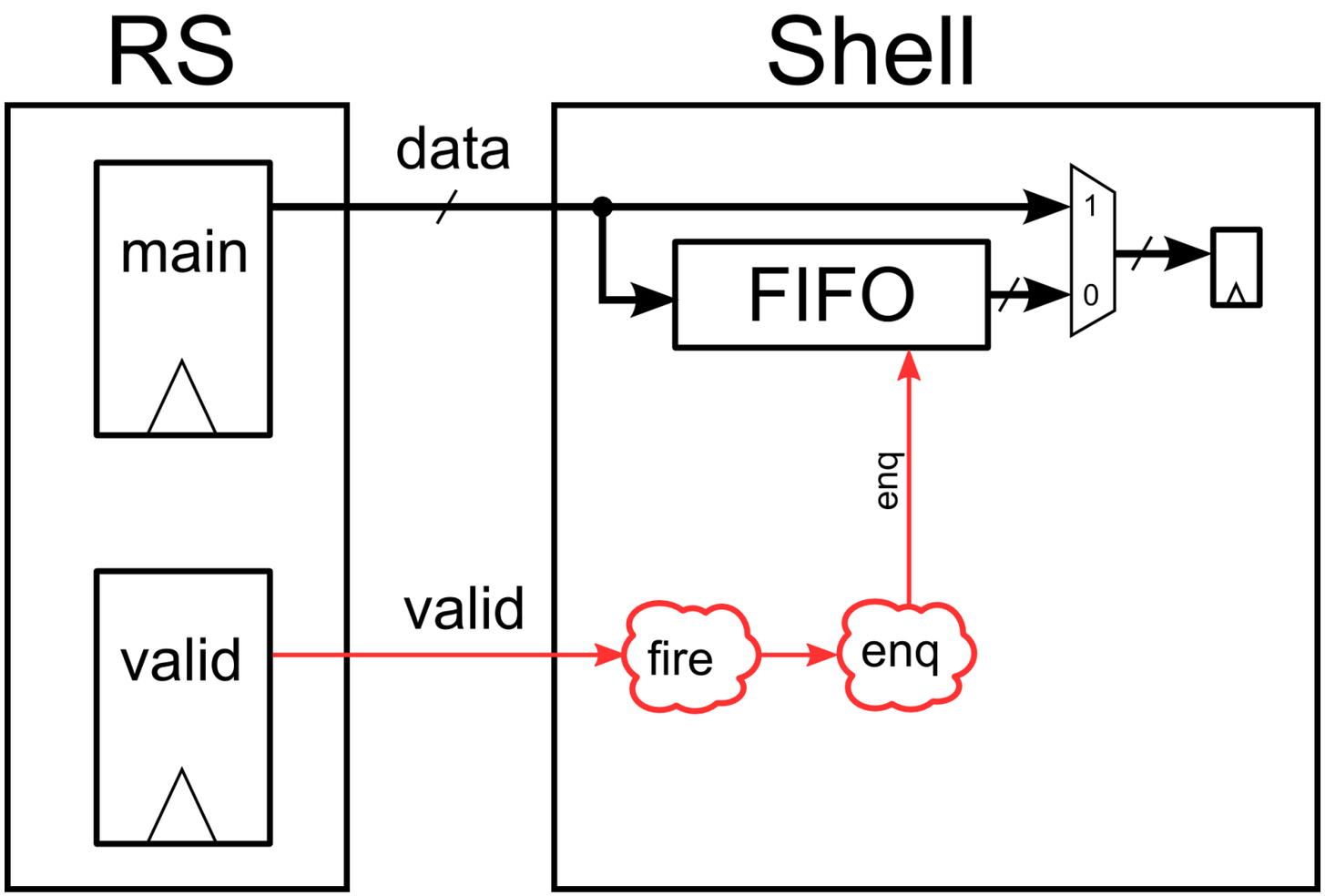


Cascaded FIR Case Study - Frequency



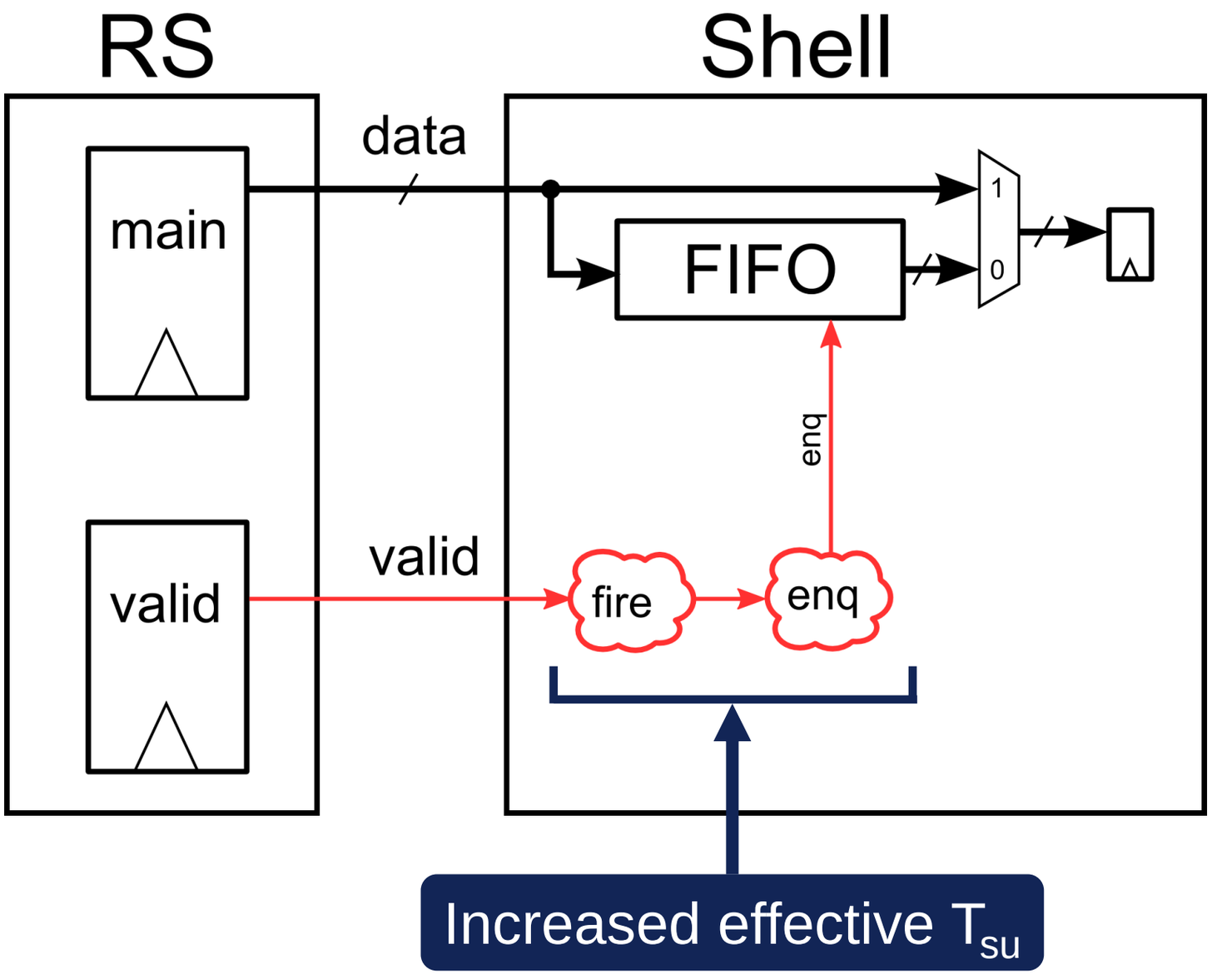
Pipelining Overhead Cause

- Extra control logic adds delay overhead to each Shell or RS



Pipelining Overhead Cause

- Extra control logic adds delay overhead to each Shell or RS



Generalized LI Scaling

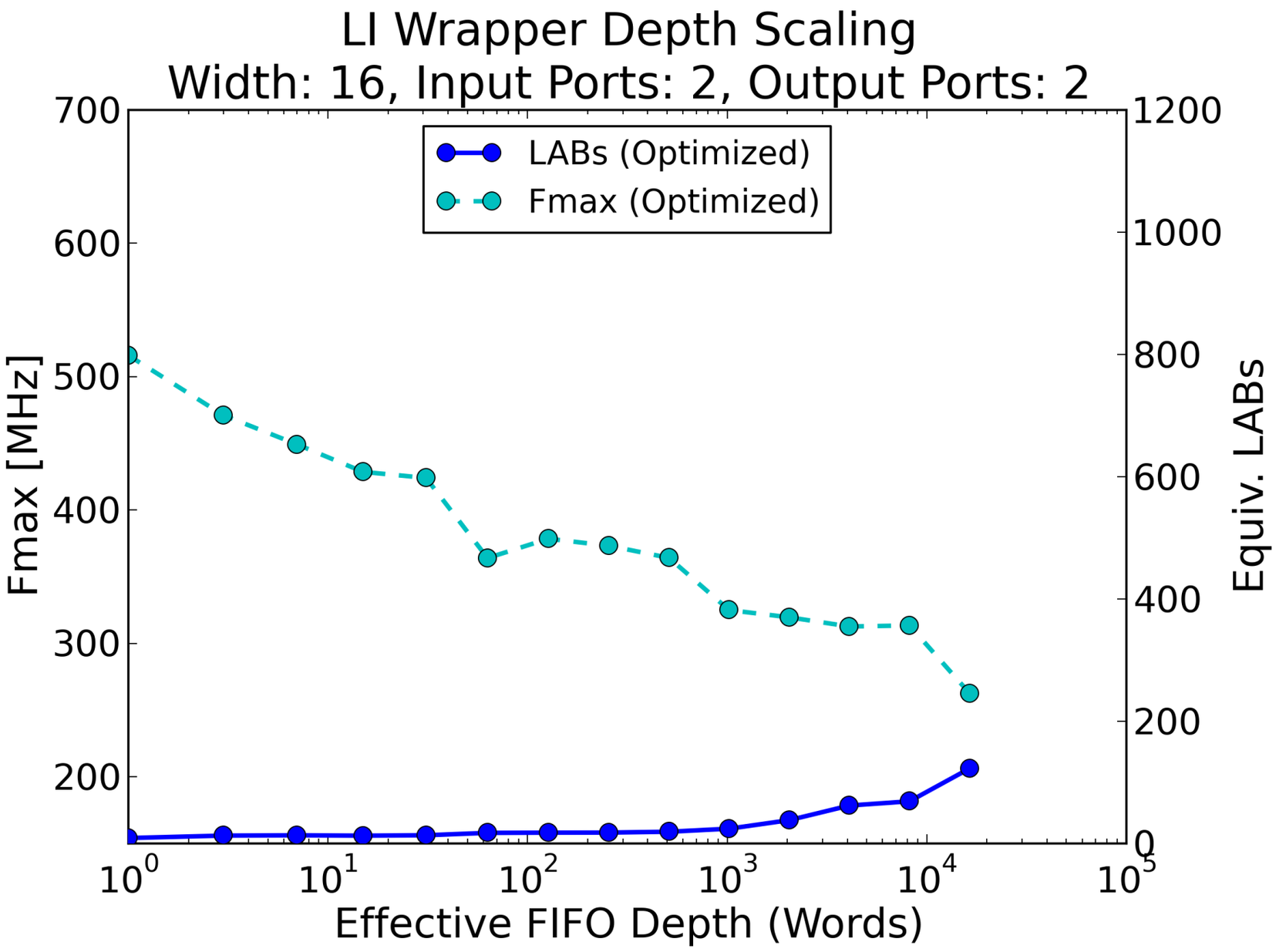
Generalized LI Shell Scaling

- Identify what makes Shells expensive
  - Leads to design guidelines to minimize overhead

- Consider impact of scaling three main shell characteristics
  - FIFO Depth
  - Number of Input Ports
  - Port Width

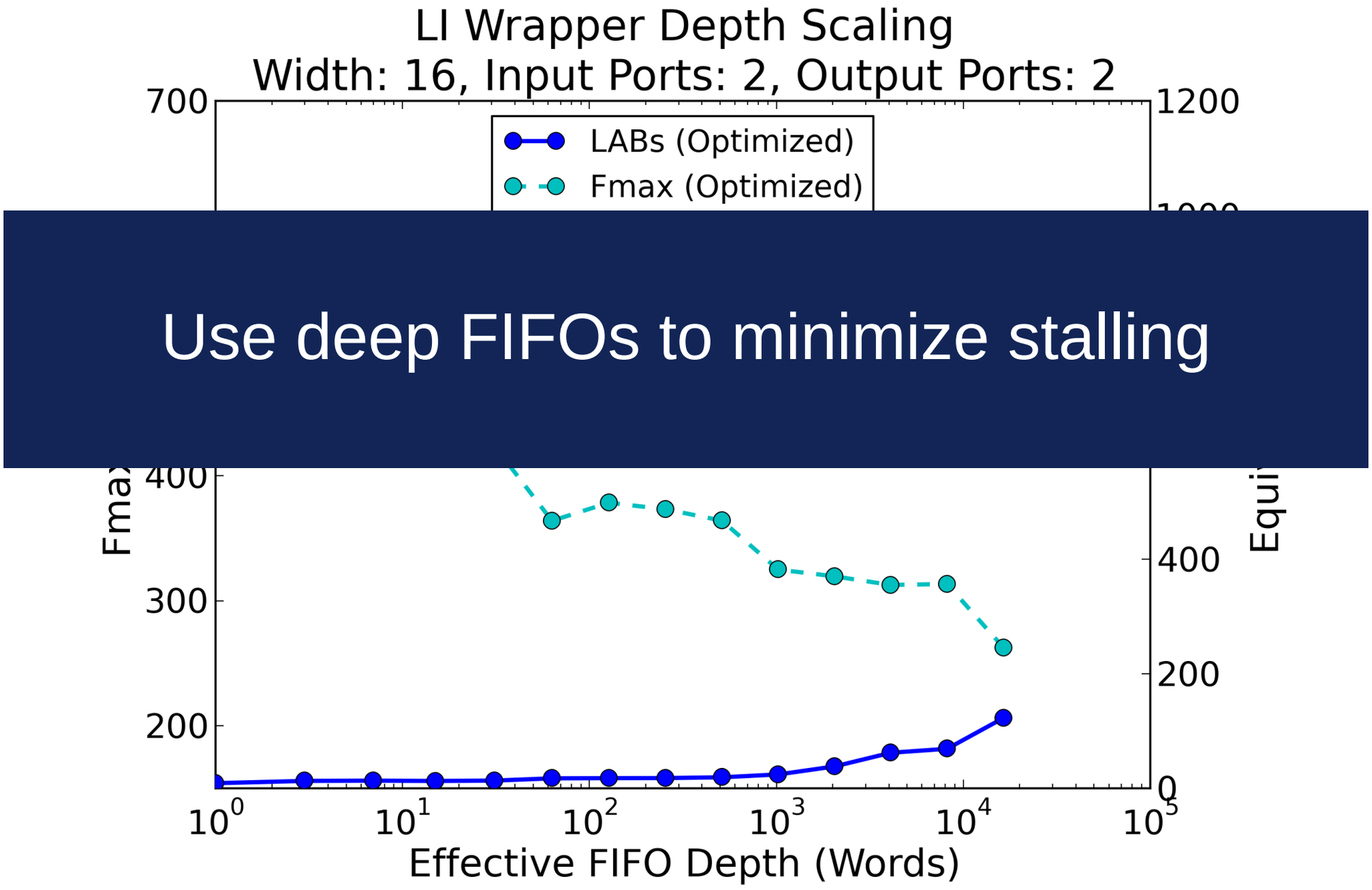
FIFO Depth Scaling

- Scaling FIFO Depth costs minimal area
  - Block RAMs are underused at shallow depths



FIFO Depth Scaling

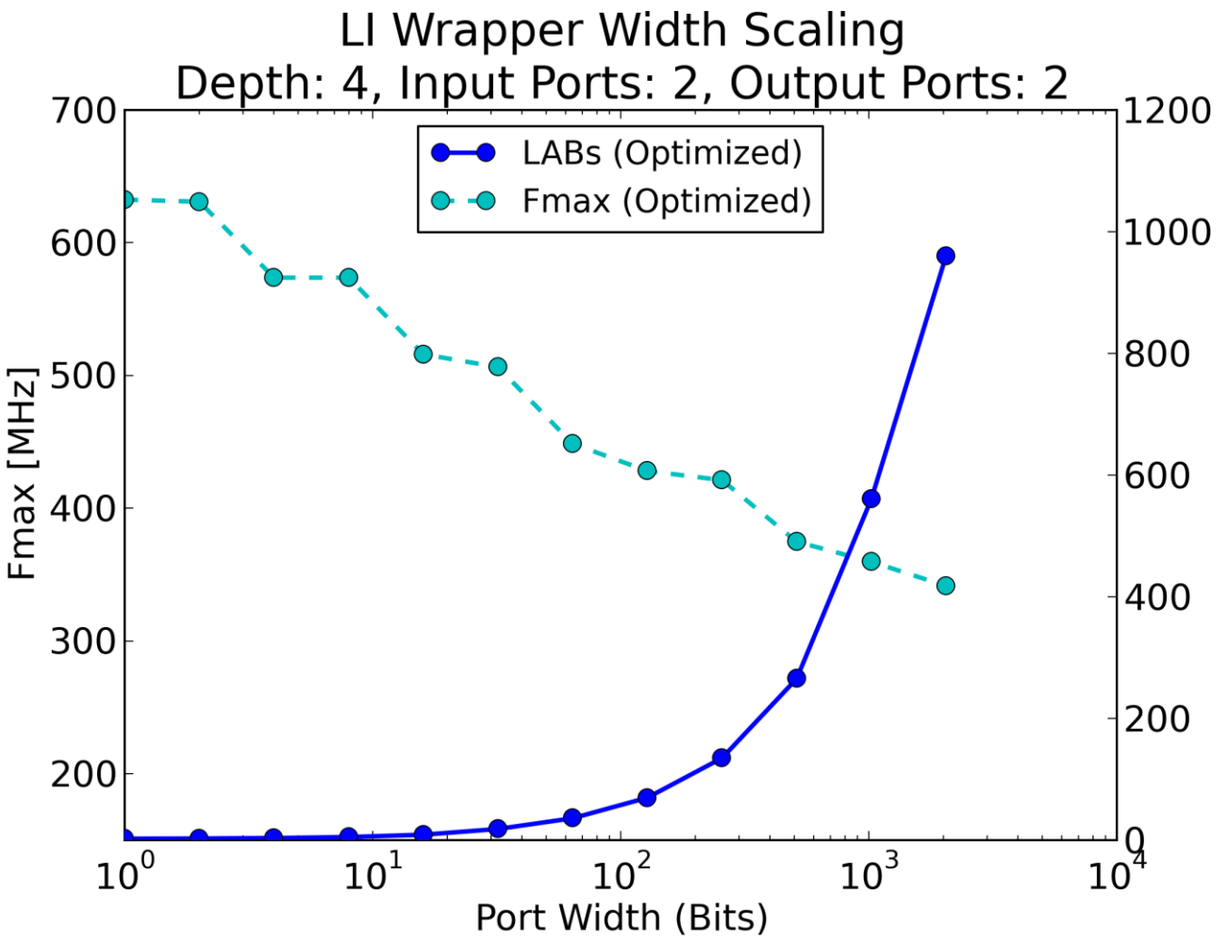
- Scaling FIFO Depth costs minimal area
  - Block RAMs are underused at shallow depths

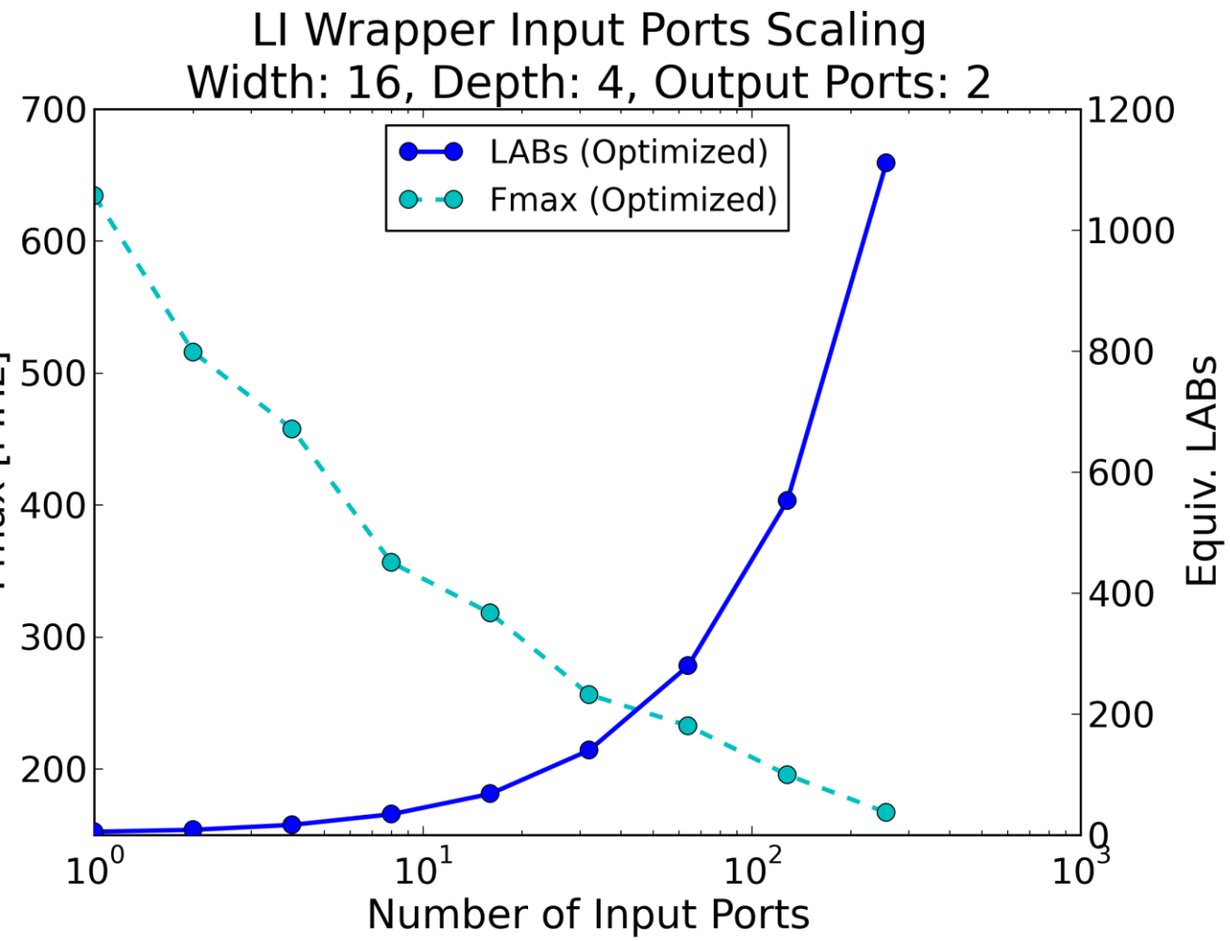


Use deep FIFOs to minimize stalling

Port Width and Input Port Scaling

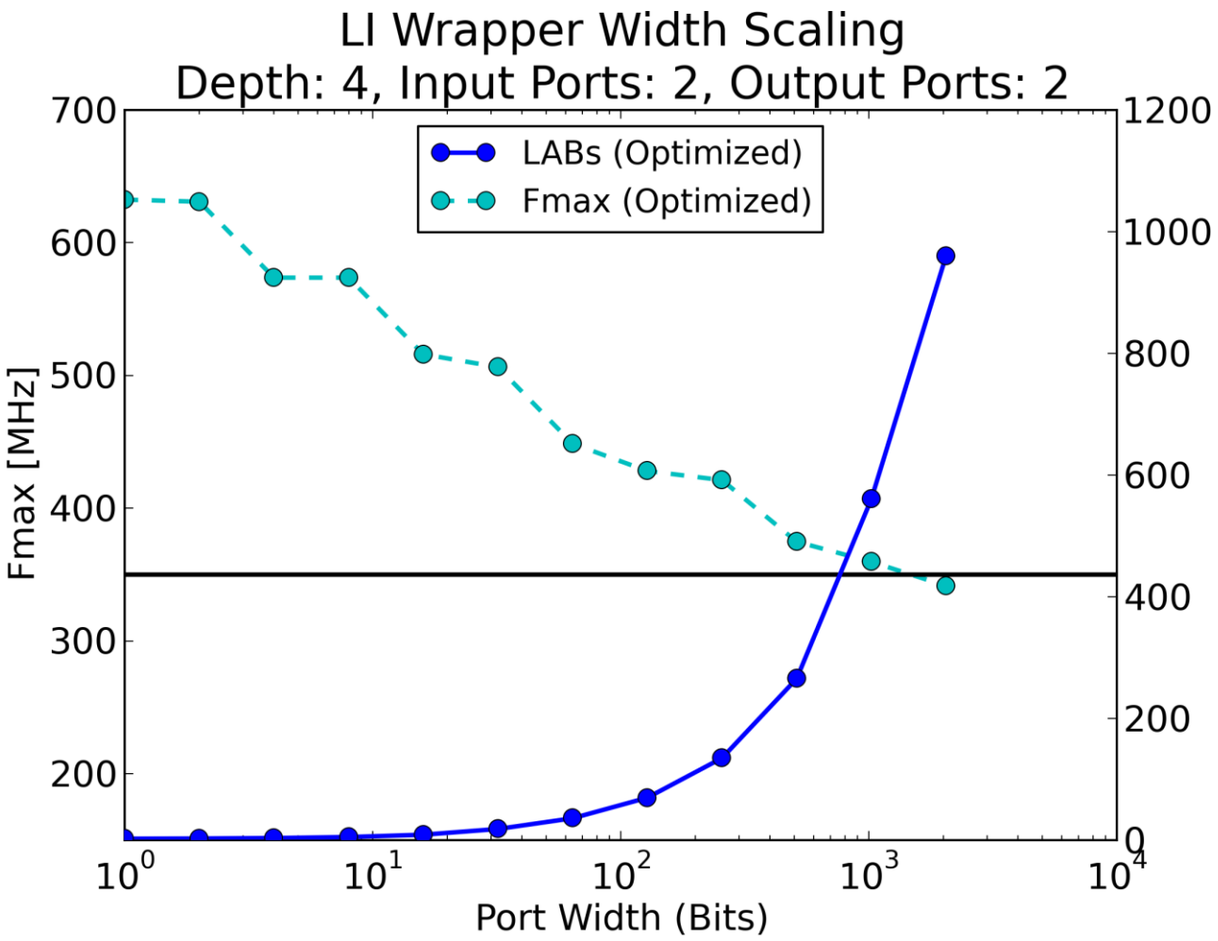
- Increasing port width or input ports costs significant area

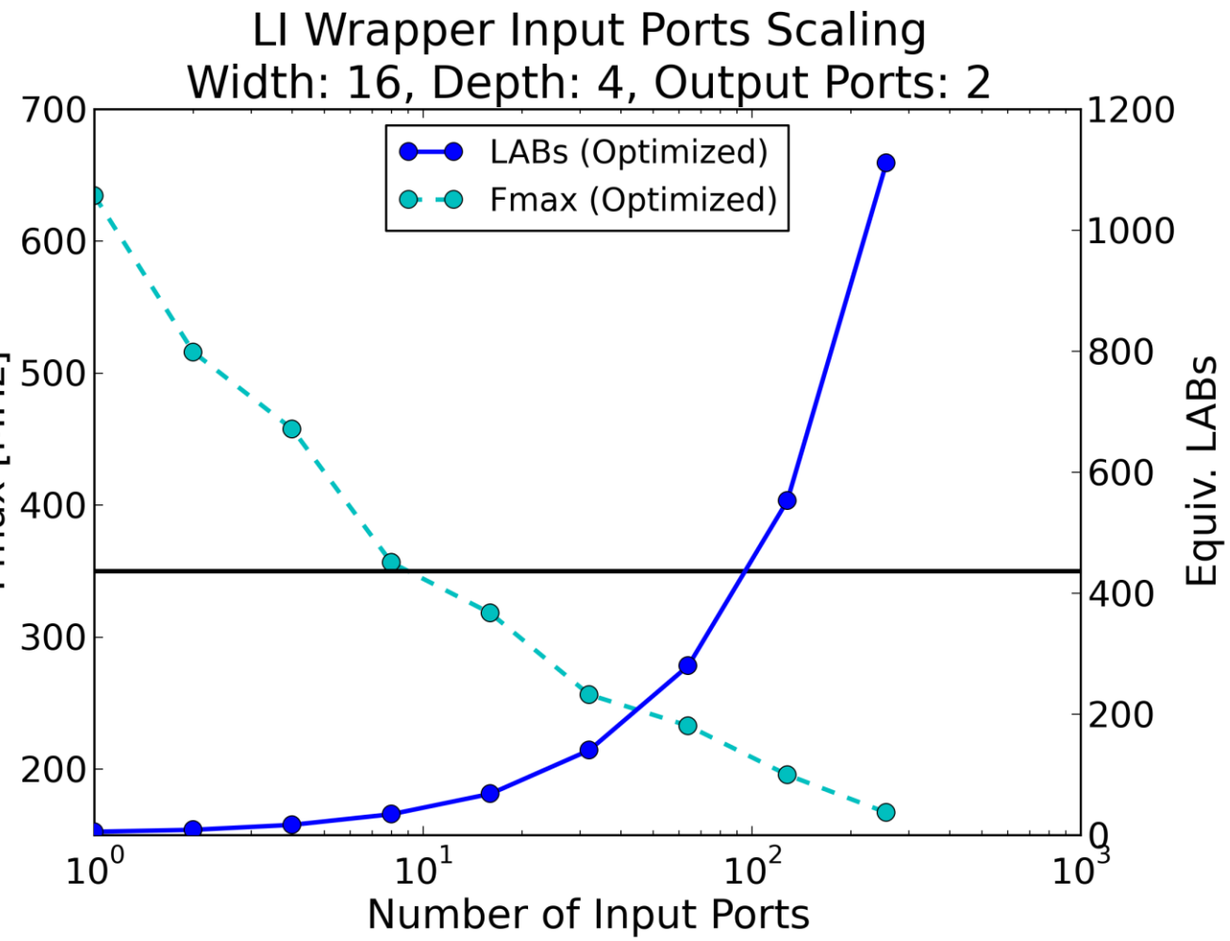




Port Width and Input Port Scaling

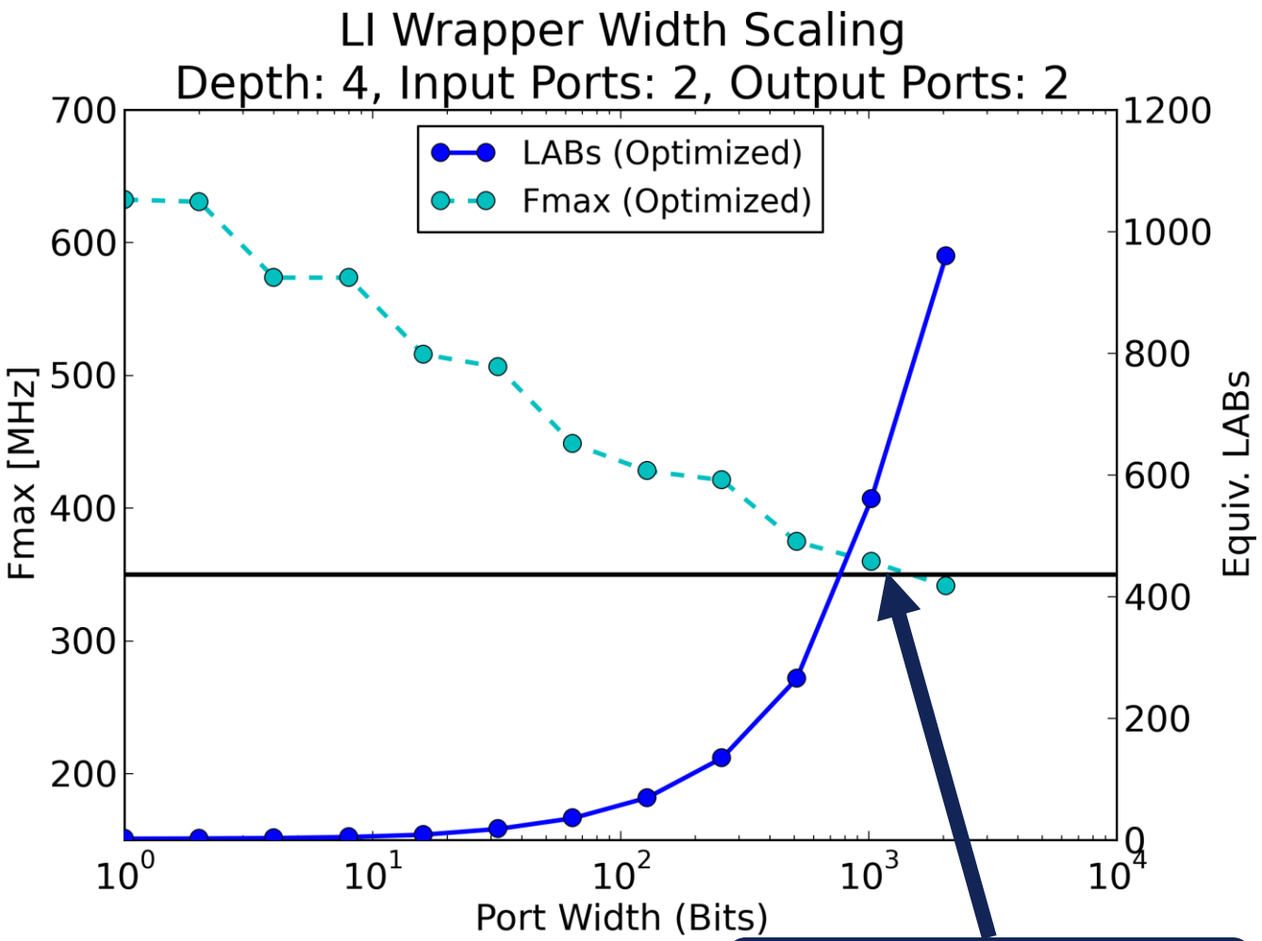
- Increasing port width or input ports costs significant area
- Frequency degrades faster as input ports are increased

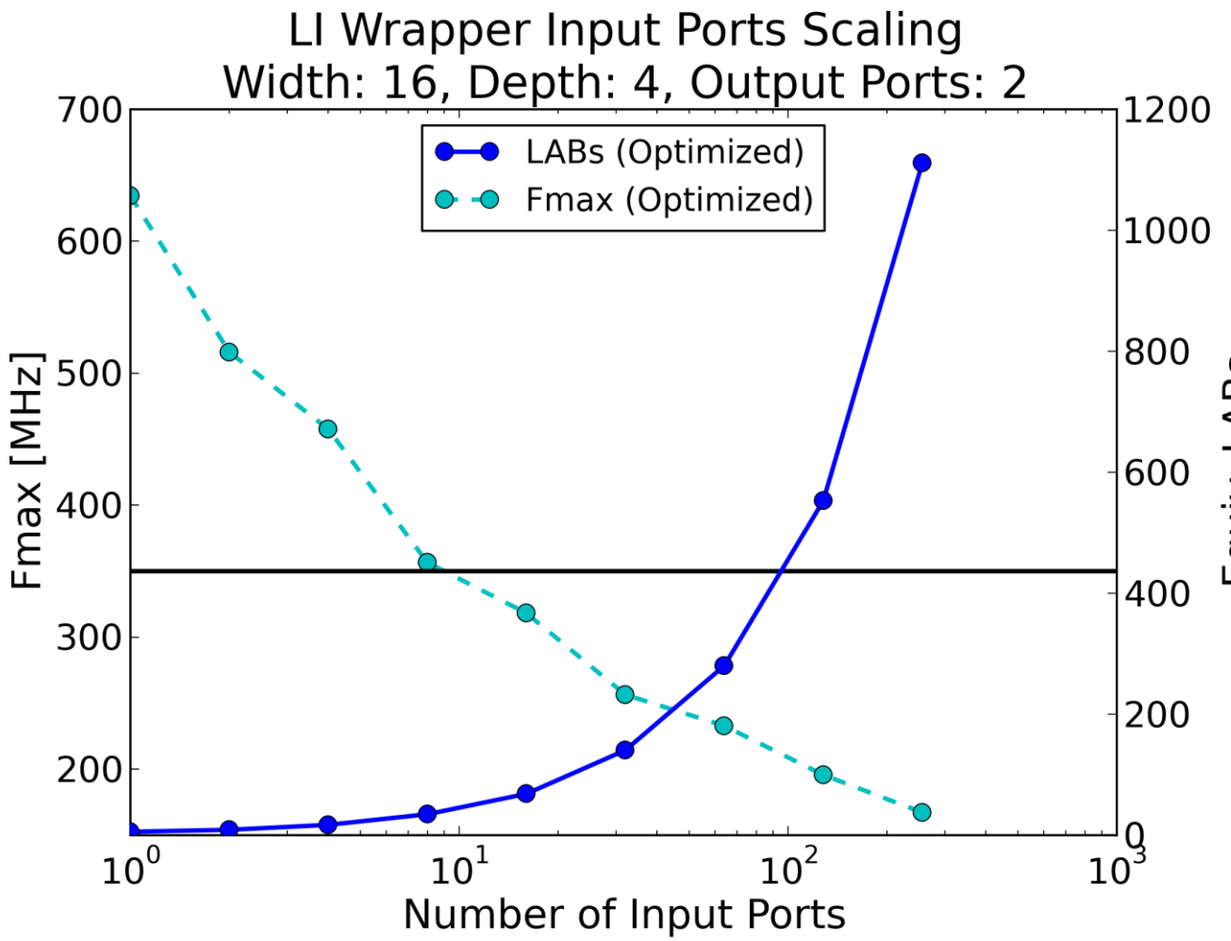




Port Width and Input Port Scaling

- Increasing port width or input ports costs significant area
- Frequency degrades faster as input ports are increased

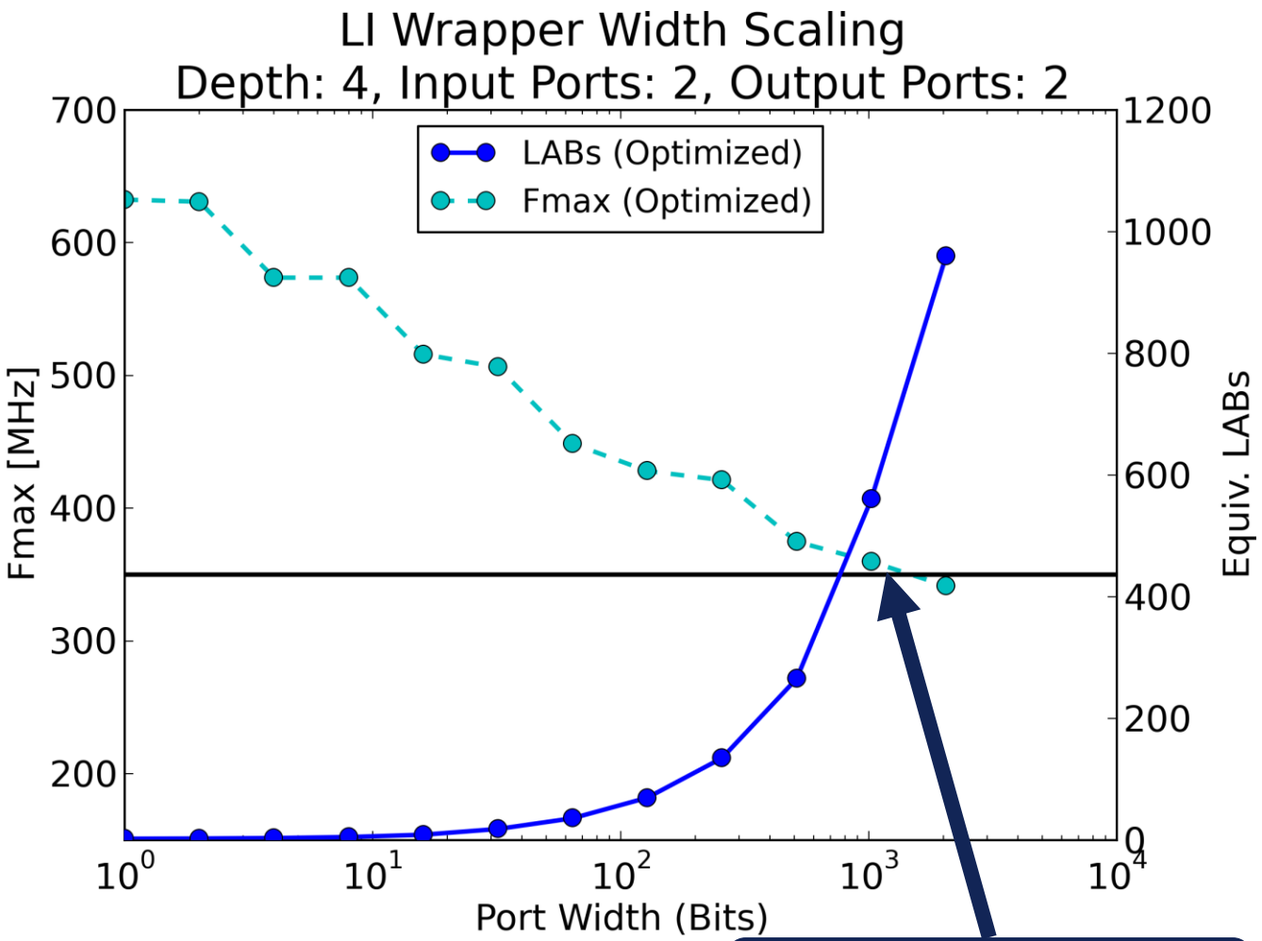


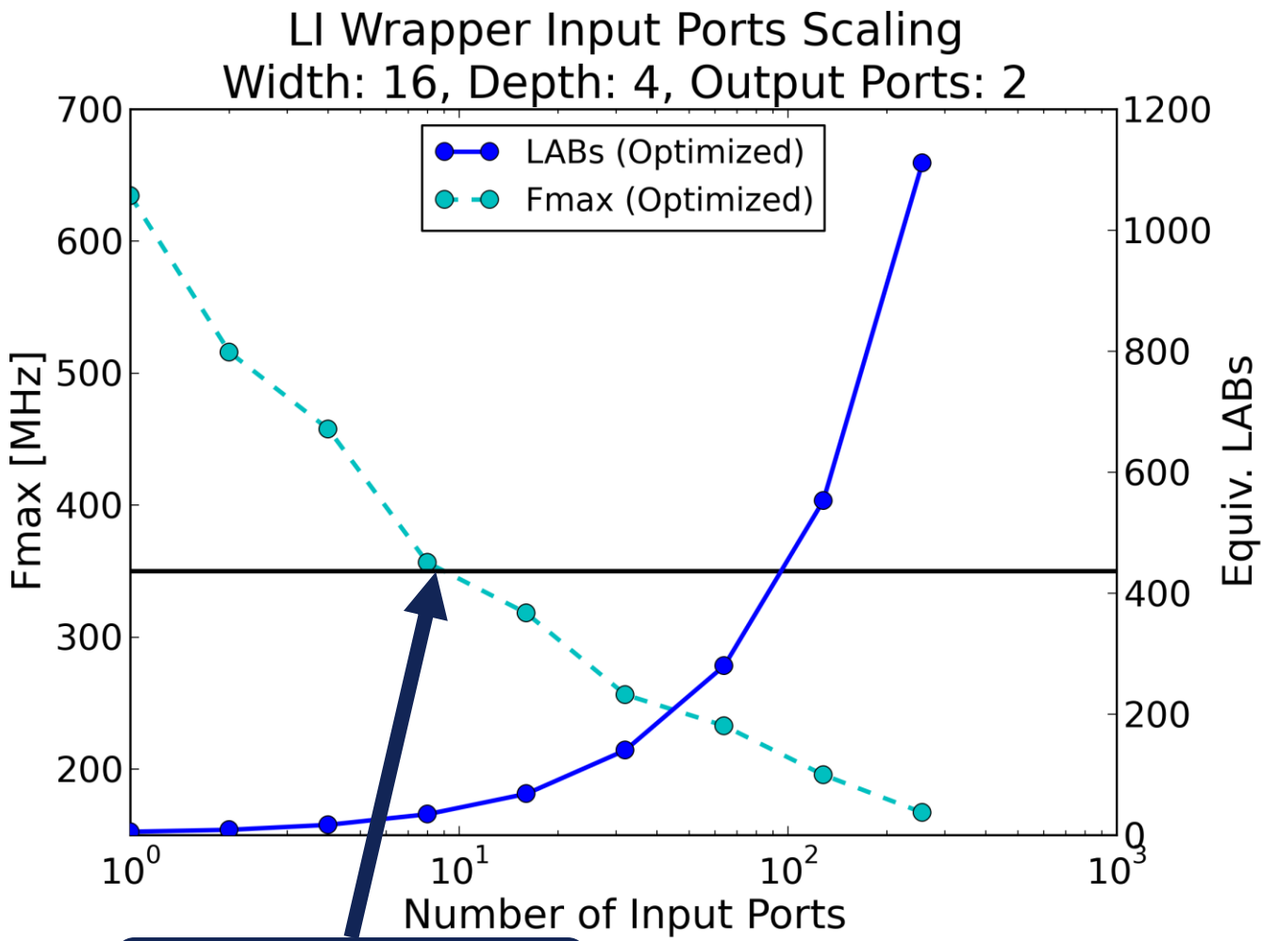


2048 input bits

Port Width and Input Port Scaling

- Increasing port width or input ports costs significant area
- Frequency degrades faster as input ports are increased



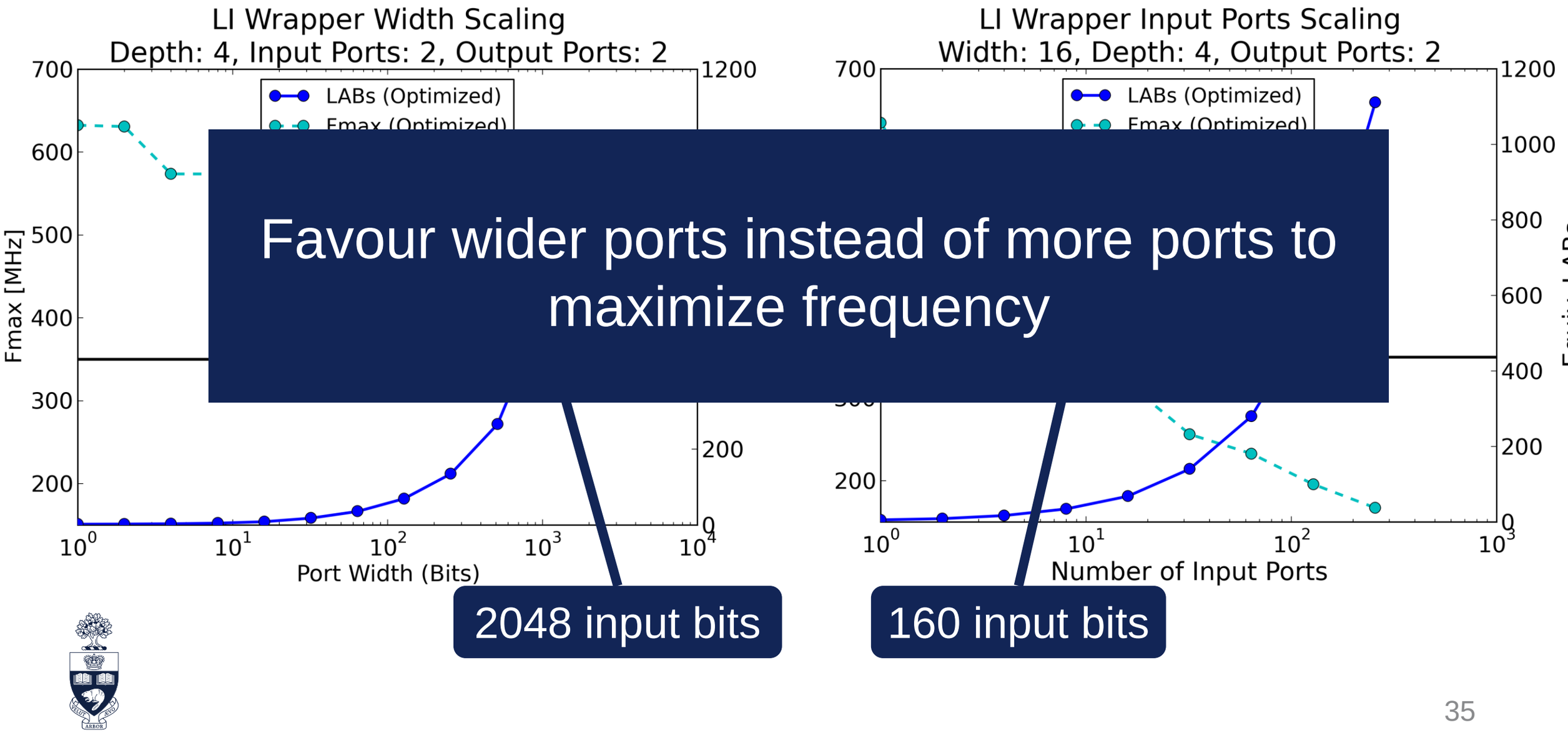


2048 input bits

160 input bits

Port Width and Input Port Scaling

- Increasing port width or input ports costs significant area
- Frequency degrades faster as input ports are increased



Granularity

LI Design Granularity

- How fine or coarse should we make LI Systems?

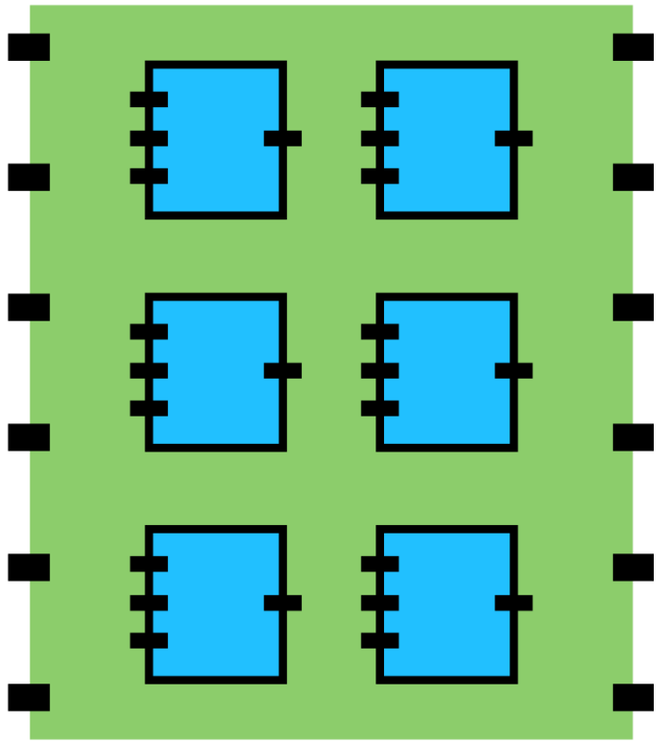
- Trade-off between:
  - Flexibility and productivity benefits
  - Area overhead

- Local communication (e.g. 40K LEs) is still fast
- Flexibility most beneficial for slow system-level communication

Rent's Rule

- Use Rent's Rule to relate design size to pin count:

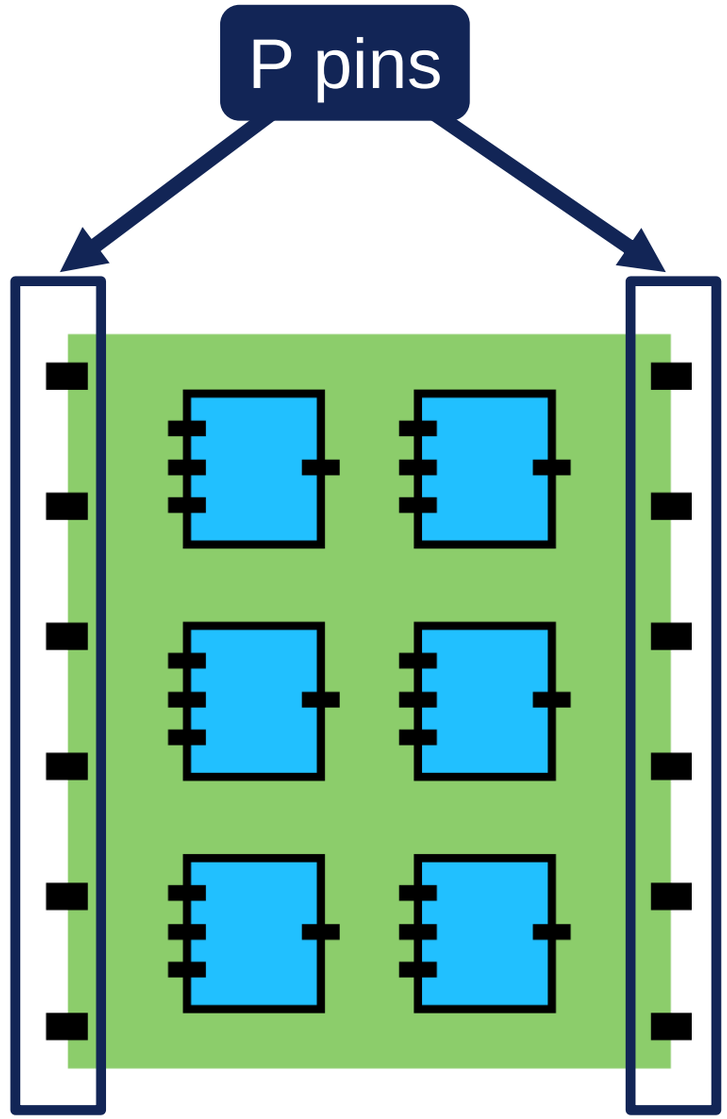
$$P = KN^R$$



Rent's Rule

- Use Rent's Rule to relate design size to pin count:

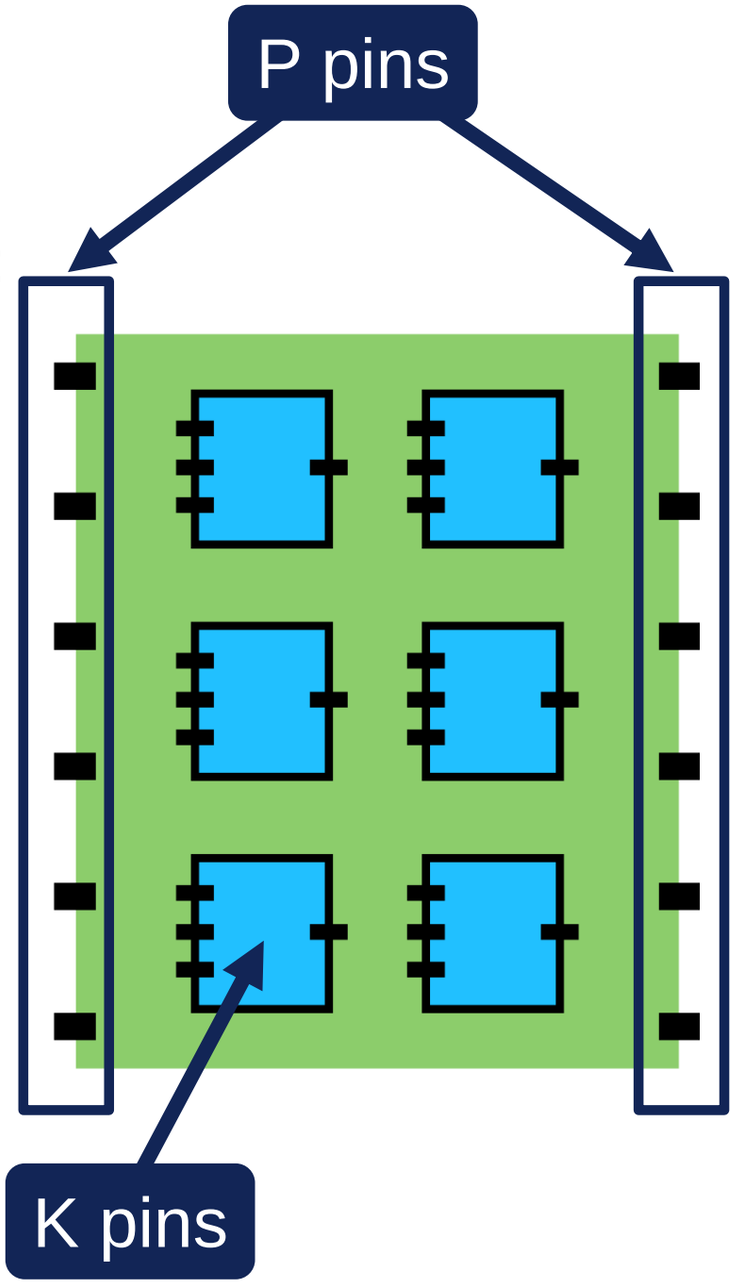
$$P = KN^R$$



Rent's Rule

- Use Rent's Rule to relate design size to pin count:

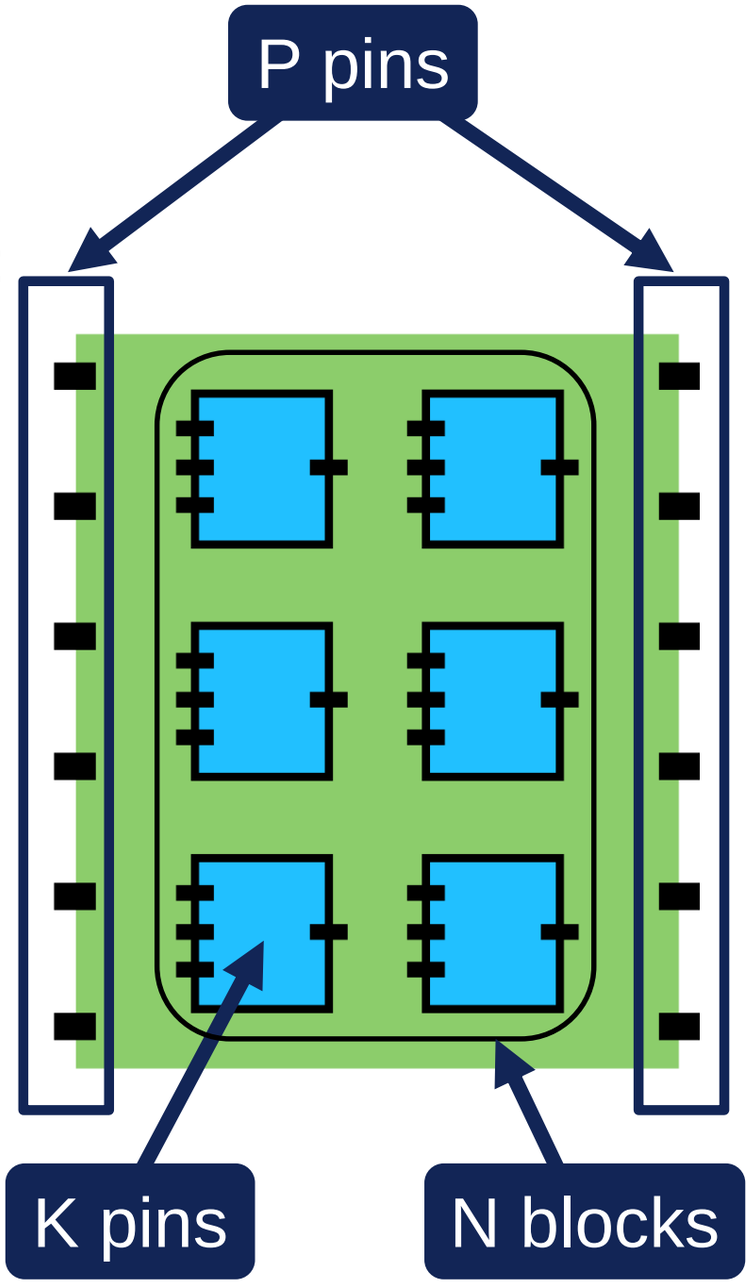
$$P = KN^R$$



Rent's Rule

- Use Rent's Rule to relate design size to pin count:

$$P = KN^R$$

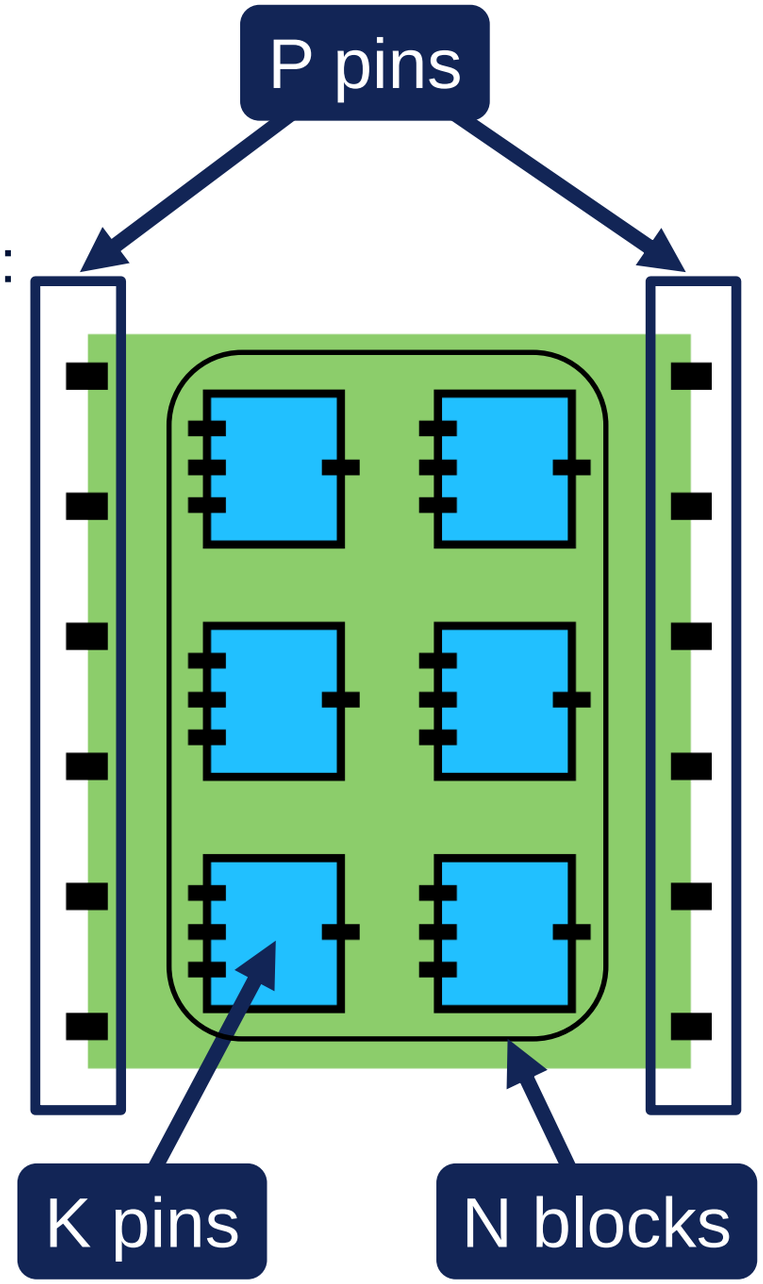


Rent's Rule

- Use Rent's Rule to relate design size to pin count:

$$P = KN^R$$

- R: Rent parameter



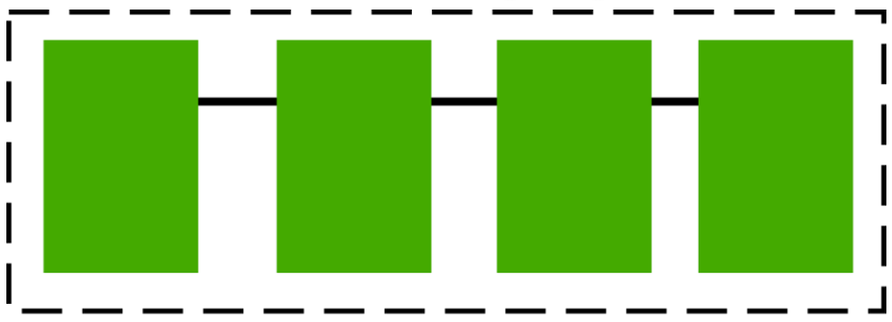
Rent's Rule

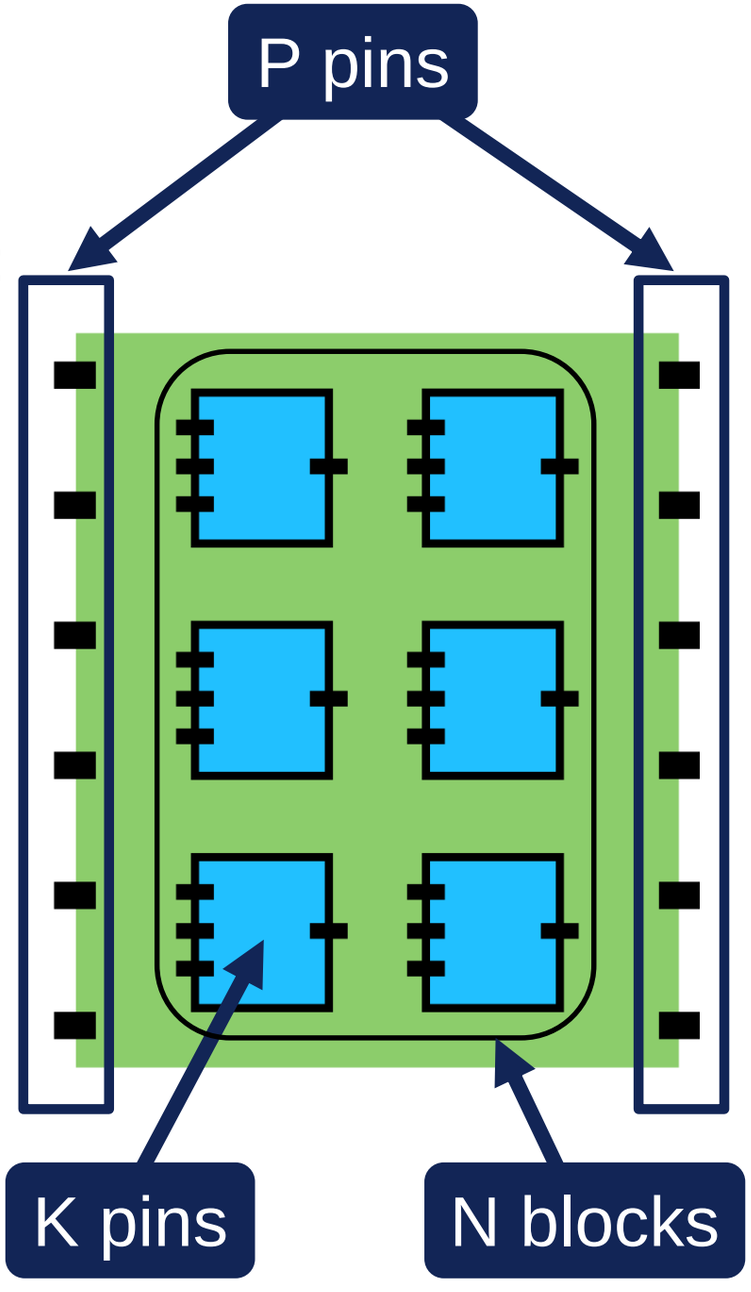
- Use Rent's Rule to relate design size to pin count:

$$P = KN^R$$

- R: Rent parameter

R = 0.0



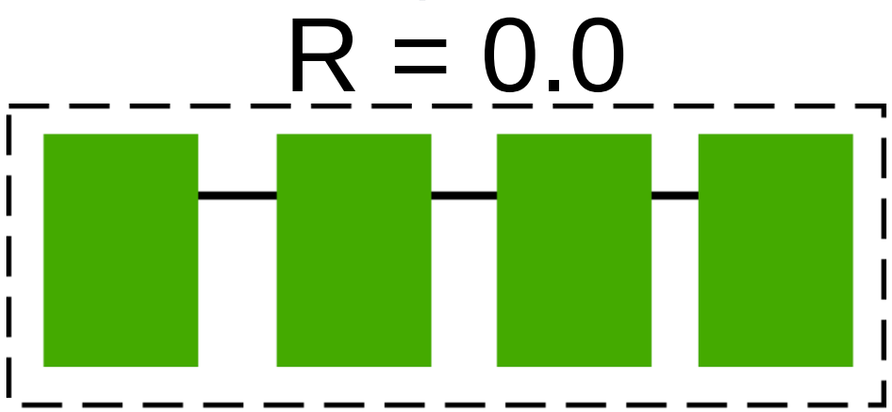


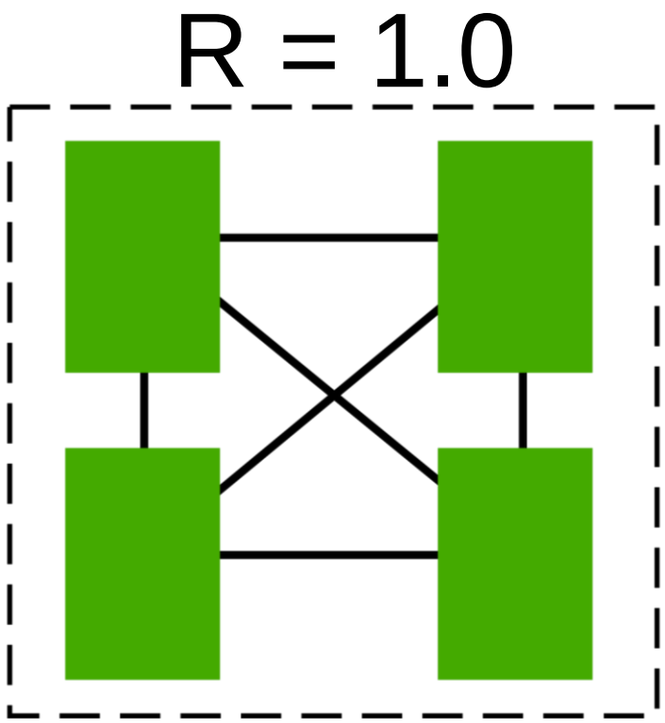
Rent's Rule

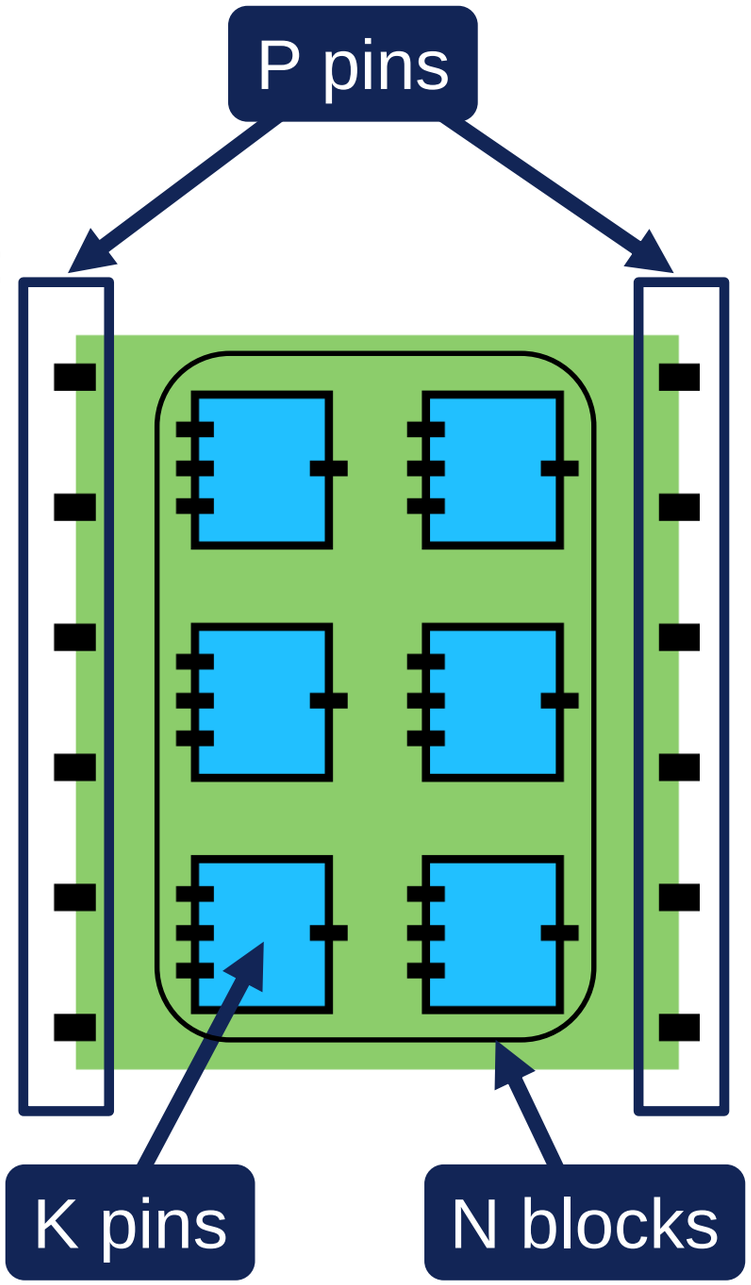
- Use Rent's Rule to relate design size to pin count:

$$P = KN^R$$

- R: Rent parameter





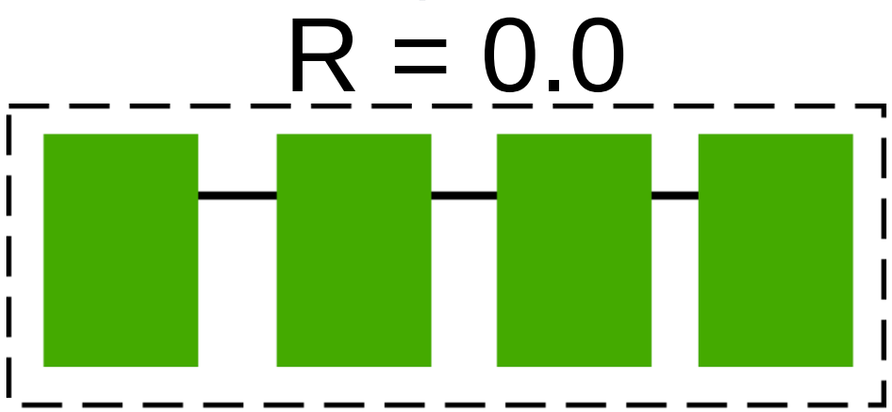


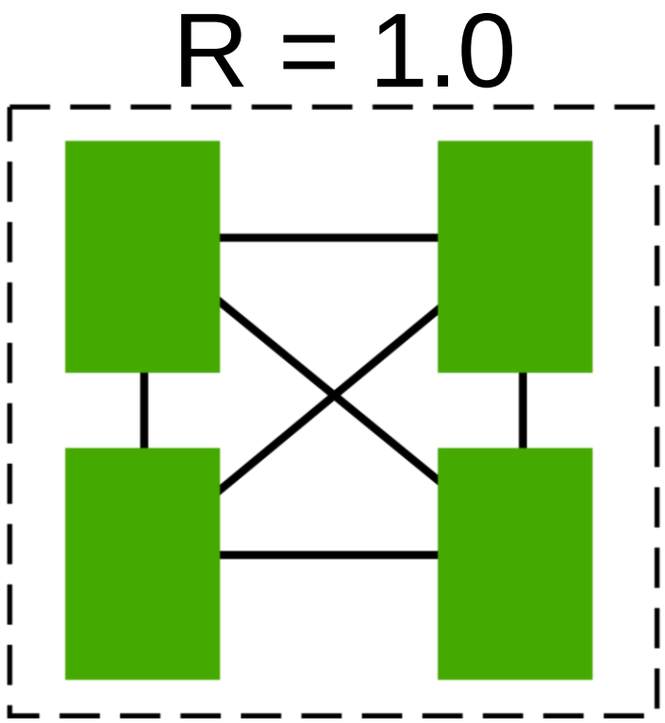
Rent's Rule

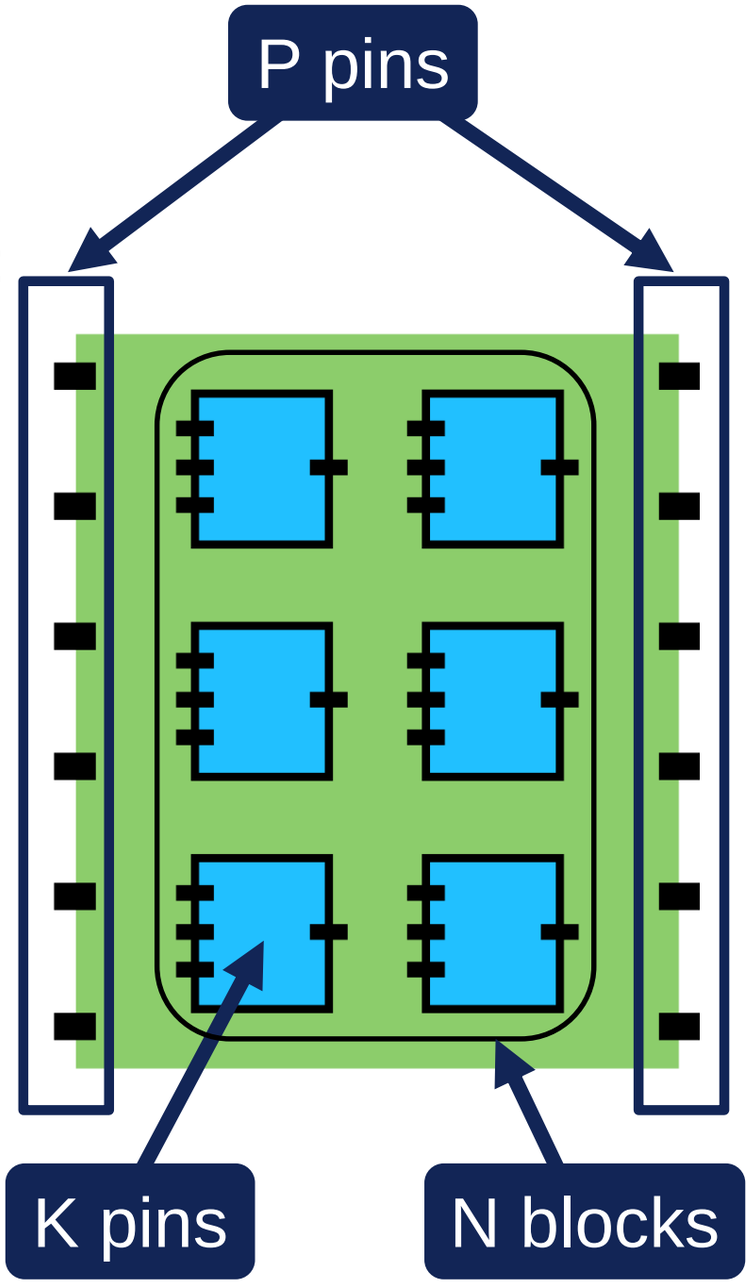
- Use Rent's Rule to relate design size to pin count:

$$P = KN^R$$

- R: Rent parameter





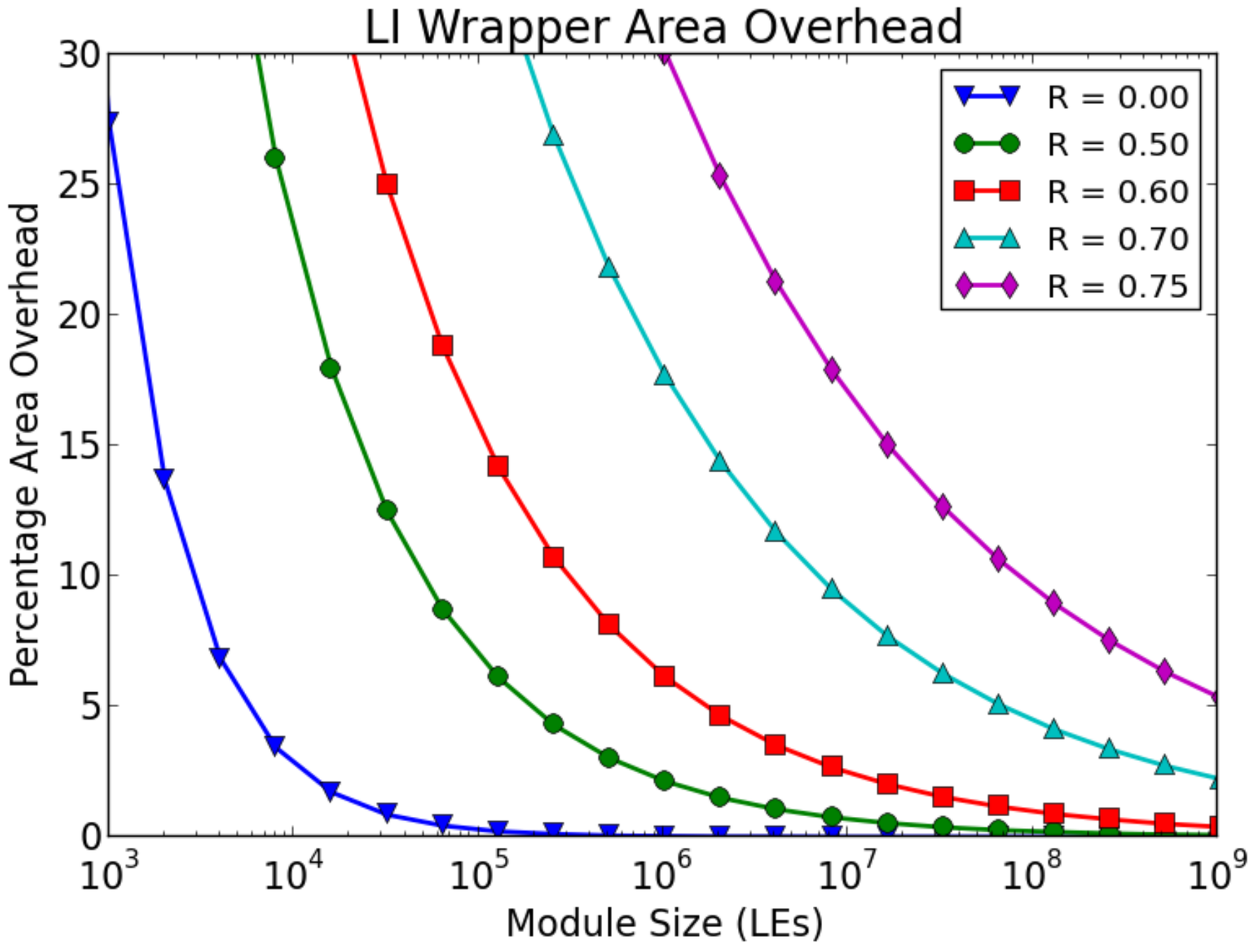


- Typical circuits:

$$0.50 < R < 0.75$$

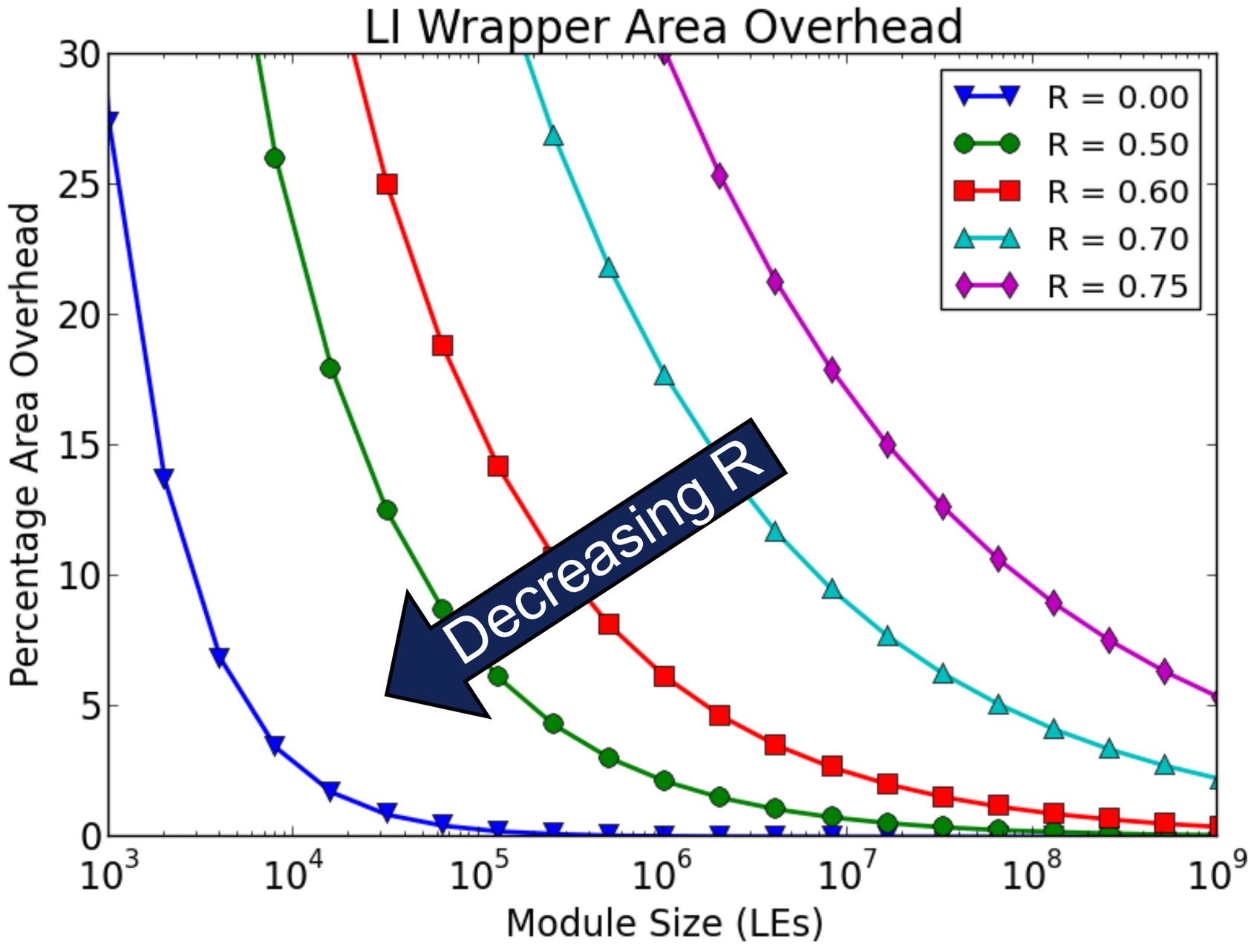
Rent's Rule Overhead Projections

- Combine shell area scaling numbers for various design sizes and ranges of Rent parameters



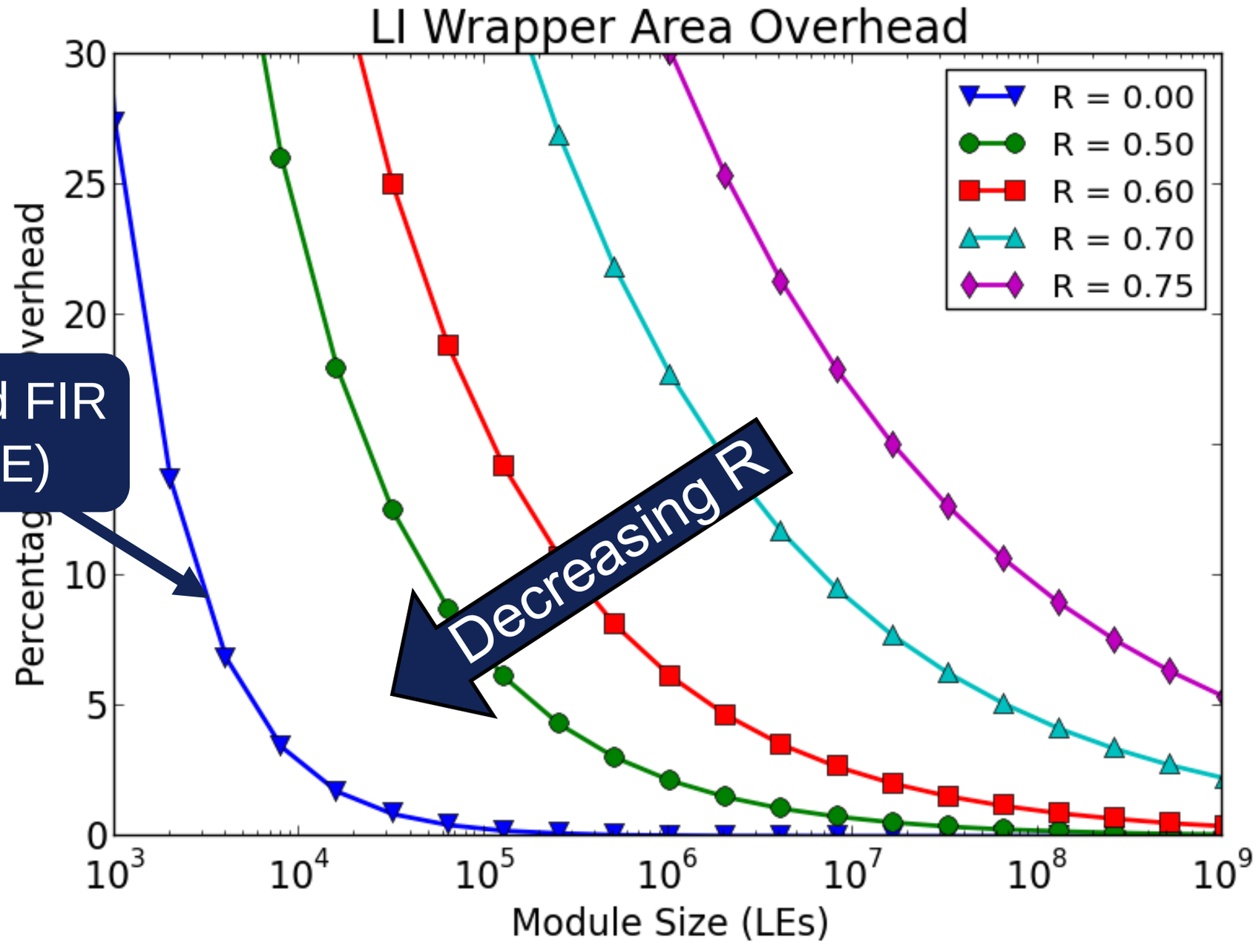
Rent's Rule Overhead Projections

- Combine shell area scaling numbers for various design sizes and ranges of Rent parameters



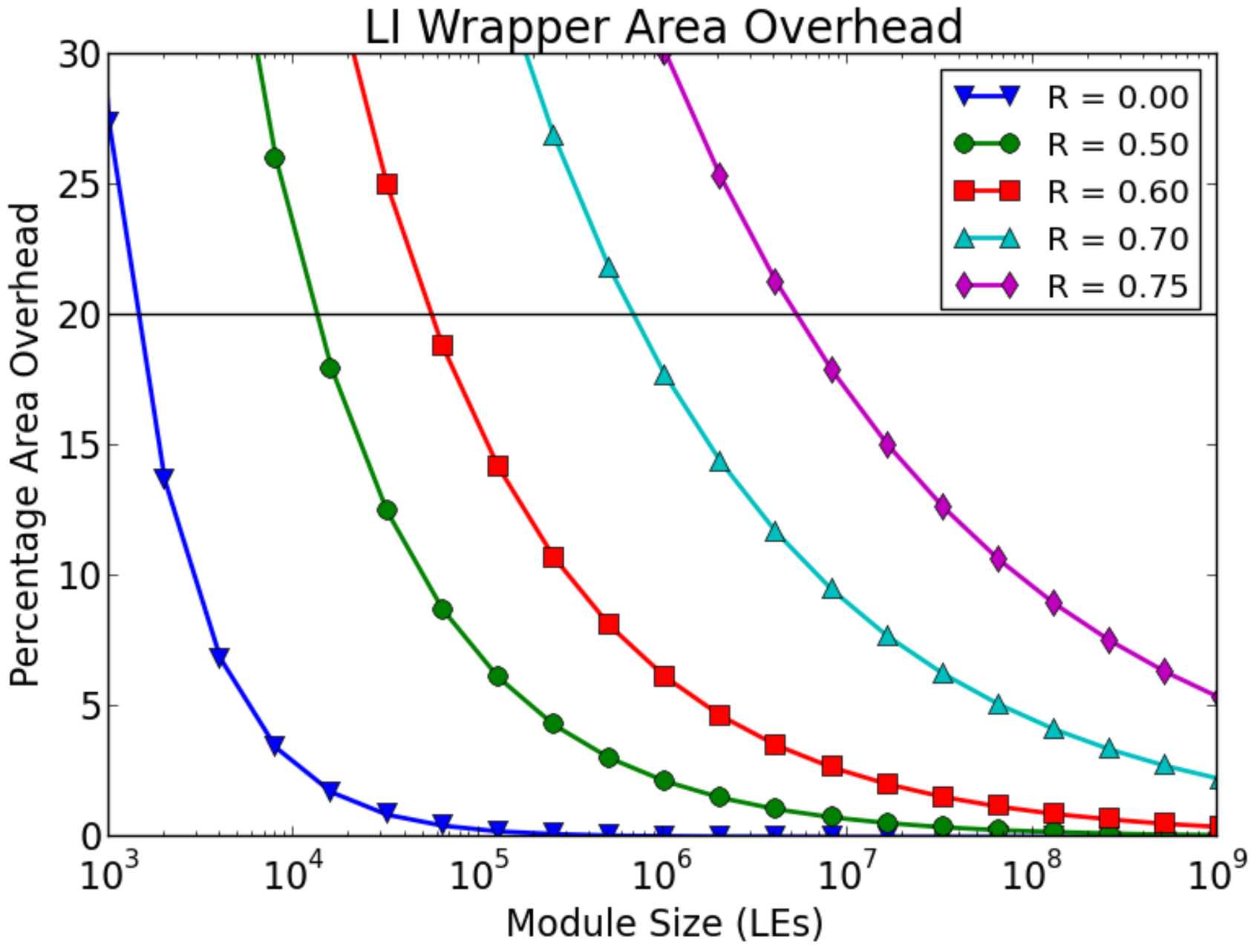
Rent's Rule Overhead Projections

- Combine shell area scaling numbers for various design sizes and ranges of Rent parameters



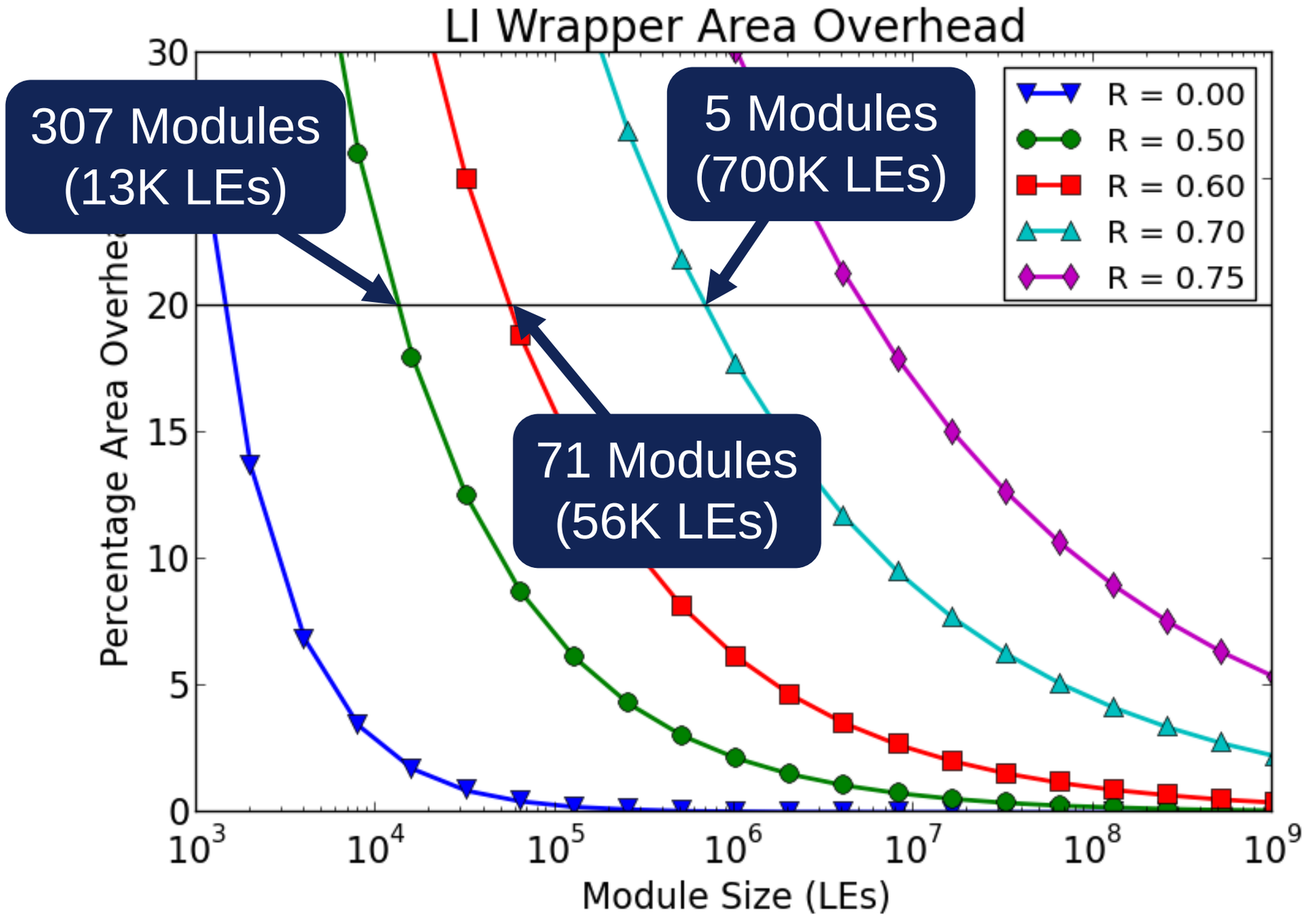
Hypothetical Design Example 20% Overhead

- Consider a 4M LE FPGA at 20% area overhead



Hypothetical Design Example 20% Overhead

- Consider a 4M LE FPGA at 20% area overhead



LI Design Granularity

- Area overhead is strongly related to communication locality (Rent Parameter)
  - Designs with well localized communication will result in low overhead

- Rent parameter varies within different parts of a design
  - Careful choice of module boundaries may further reduce overhead

Conclusion and Future Work

Conclusion

- Illustrated the growing gap between local and global communication speed
  - 3.6x and growing

- Developed optimized LI building blocks for FPGAs
  - Reduced frequency overhead from 33% to 8%

- Quantified the area and frequency overhead of LI communication on FPGAs

- Provided design guidelines to minimize the overheads of LI communication

Future Work

- Explore the benefits of LI design
  - LI aware CAD Tools

- Investigate architectural enhancements
  - Hardened FIFOs
  - Fine-grained clock gating
  - Embedded NoC

- Evaluate LI design on a broader range of designs

- Develop lower area/speed overhead LI design techniques

Thanks!

Questions?

Email: kmurray@eecg.utoronto.ca

UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE & ENGINEERING