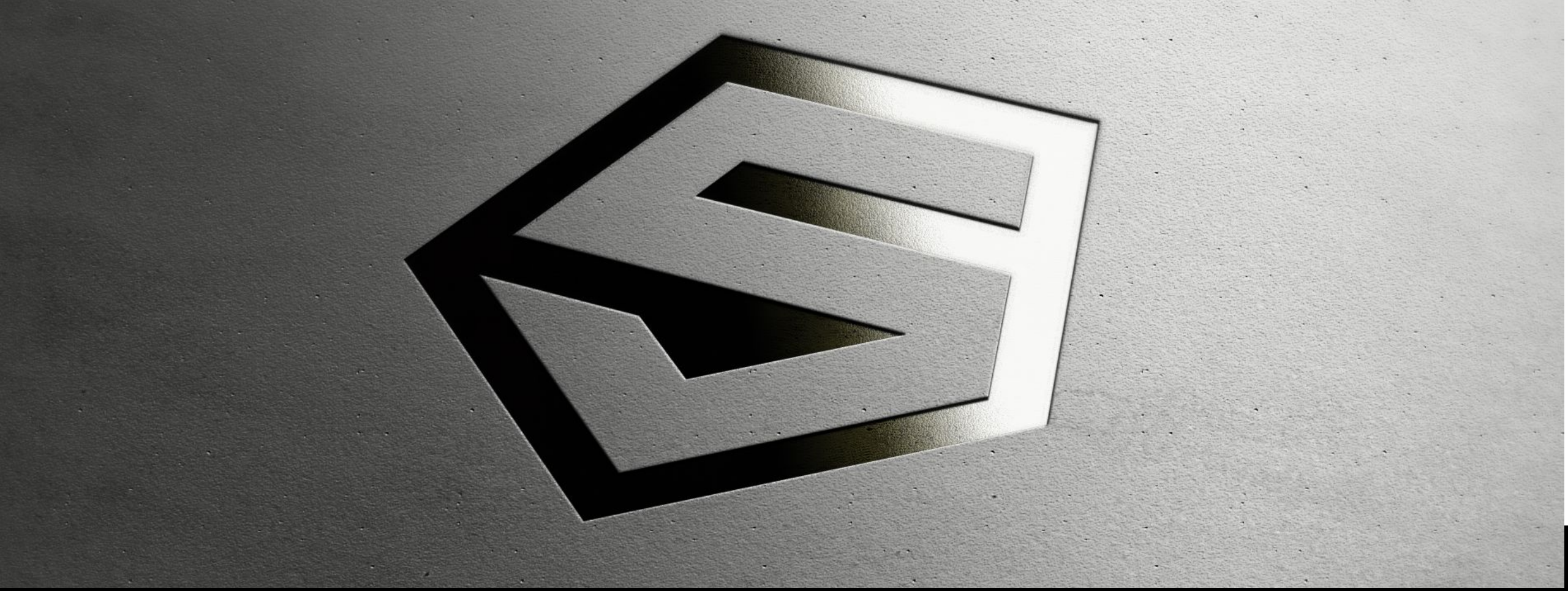


Diplomatic Design Patterns

A TileLink Case Study

Henry Cook
Wesley Terpstra
Yunsup Lee

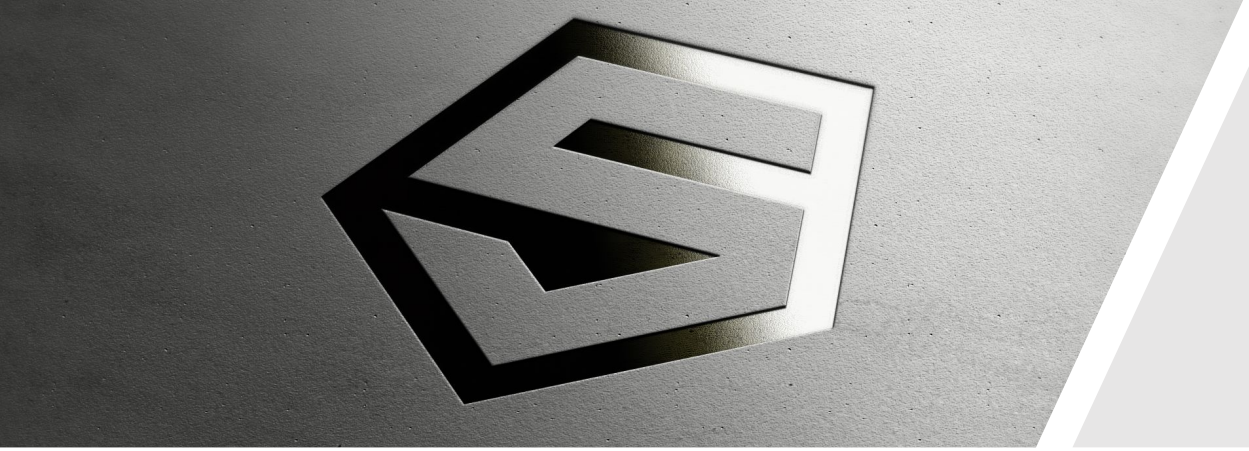
10/14/2017

Agenda

- Rocket-Chip Ecosystem
- Diplomacy
- TileLink
- Design Patterns
  - DRYing out Parameterization
  - Generation with a View
  - Compositional Cake
- Deployment

The Rocket-Chip Ecosystem for RISC-V

- Chisel
  - A Domain-Specific Language for hardware construction embedded in Scala
- Rocket-Chip
  - A collection of hardware generators implemented in Chisel
- Diplomacy
  - A Scala framework for negotiating parameters between Chisel generators
- TileLink
  - A parameterized chip-scale shared-memory interconnect protocol



Diplomacy

When Chisel Isn't Enough

- We'd like to be able to see everything that will be in the system before we even start to emit any hardware descriptions

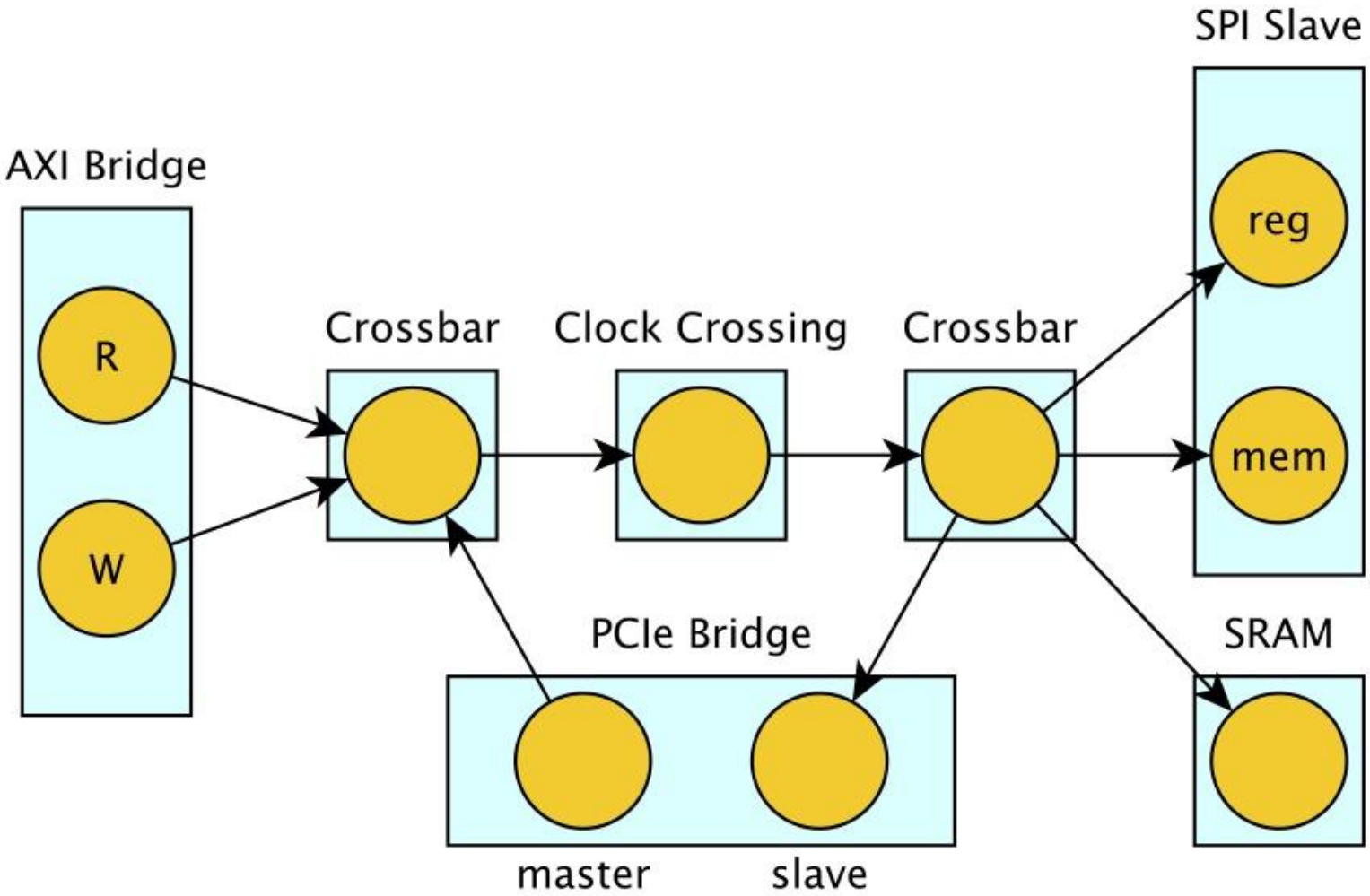
- How early can we detect misconfiguration?
  - <> considered harmful
  - wire width inference can mask problems

- Generators are nice...
  - but parameterization itself is a source of complexity
  - but each generator needs to agree with the others about what to generate

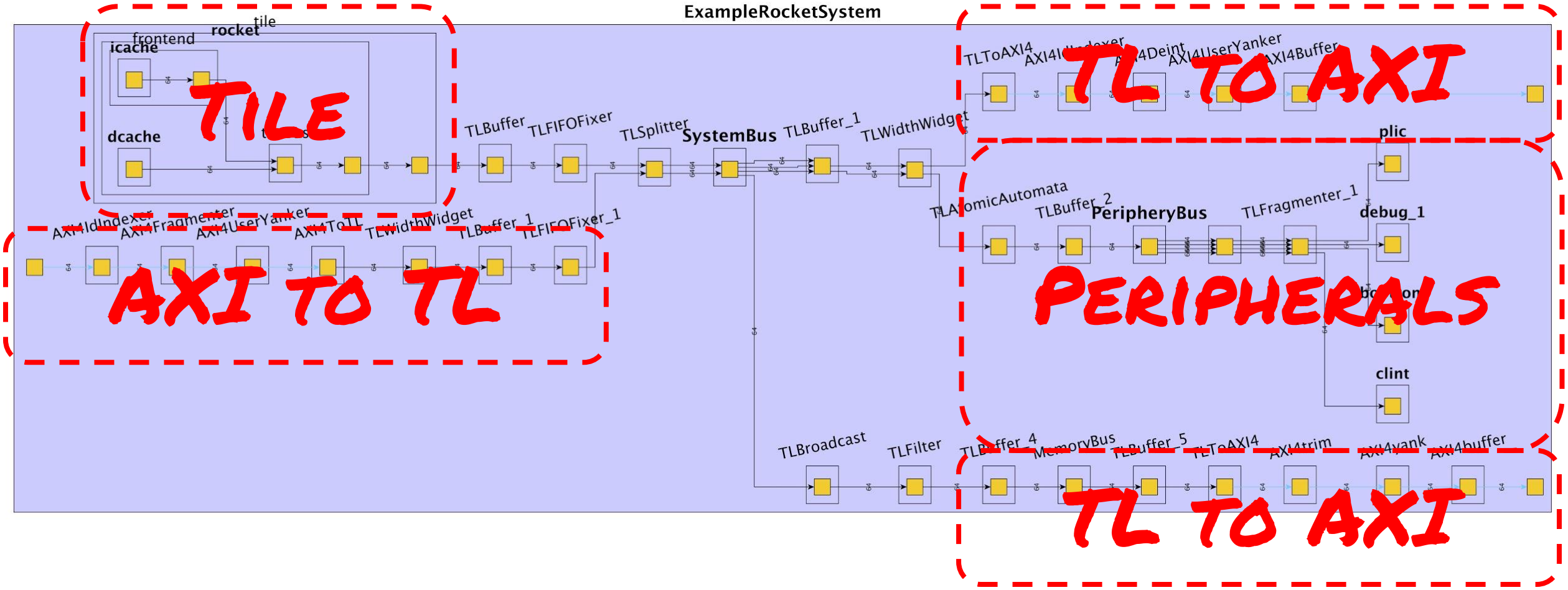
Diplomatic Parameterization

- Graphical abstraction of interconnectivity
  - Directed Acyclic Graph (DAG)
- Node
  - point where parameterized hardware is going to be generated
- Edge
  - a directed pairing between a master and slave interface
- Modules may have many nodes and nodes may have many edges

Simple Example Graph



Actual (small) Rocket-Chip Graph



SiFive

What Parameters Can Be Diplomatic?

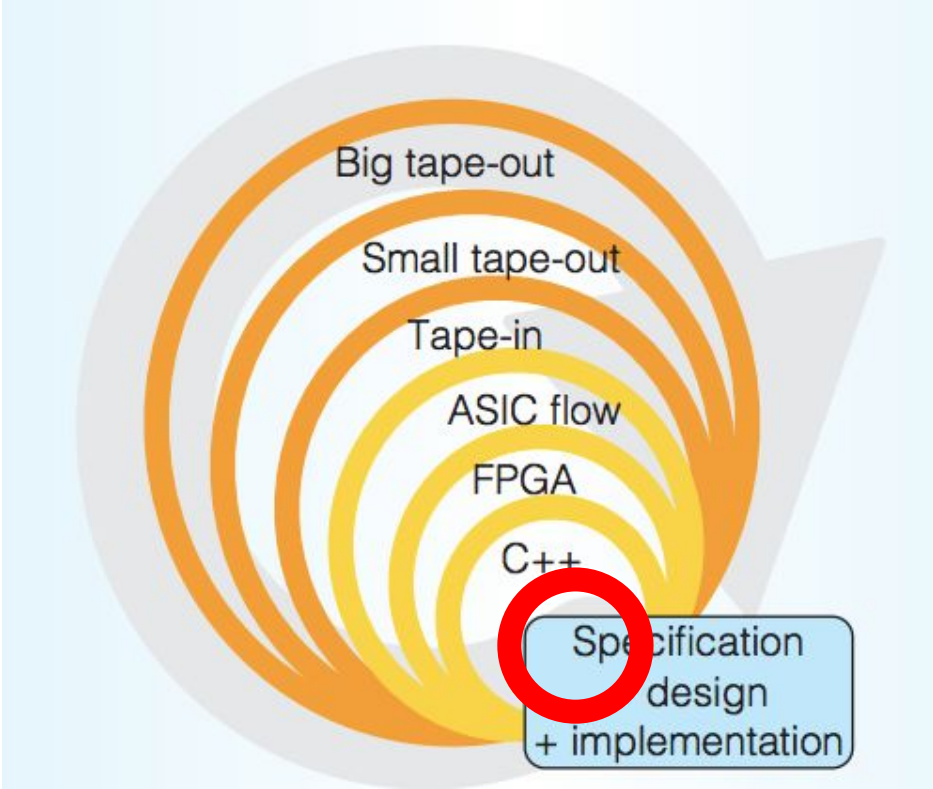
- The cardinality of sources connected to a particular sink (transitive)
- The cardinality of sinks connected to a particular source (transitive)
- The type and size of operations issues by each master and slave
- The type and size of operations allowed on a particular address
- Other properties governing allowed behavior on particular regions (e.g. modifiability, executability, cacheability)
- Ordering requirements on operations over subgraphs (e.g. FIFO)
- Presence of certain fields within control wire bundles
- Widths of fields within with control or data wire bundles
- Presence of entire virtual networks or channels

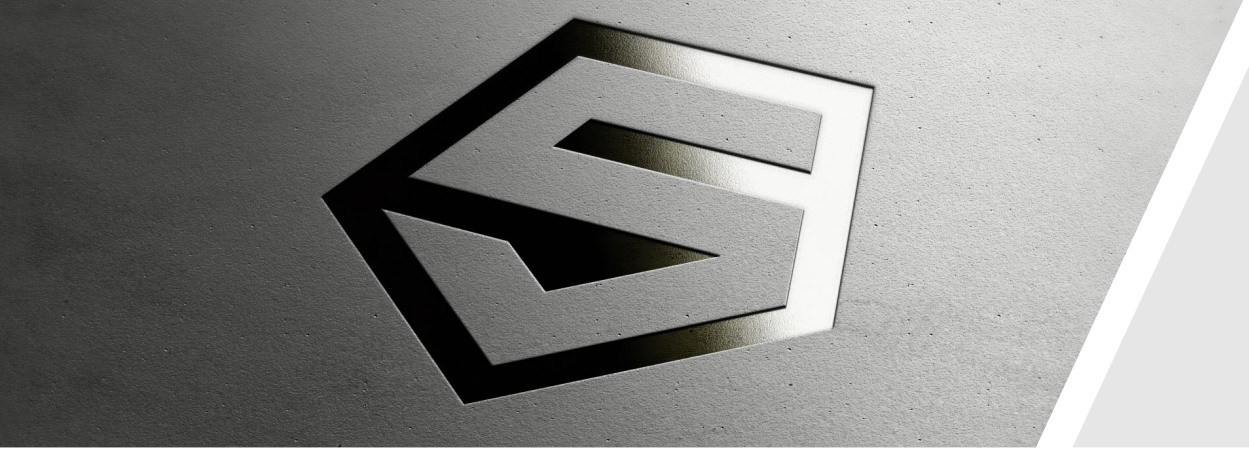
How Diplomacy Negotiates

- Two-phase elaboration
  - First, graph is created and parameters are negotiated (LazyModules)
  - Second, Chisel hardware is elaborated (Module implementations)
- During the first ("diplomatic") phase:
  - One set of parameters flow outwards from master to slave nodes
  - One set of parameters flow inwards from slave to master nodes
  - Adapter nodes can modify parameters flowing through them
- All modules can place requirements on parameter values

Short Circuiting the Debug Loop

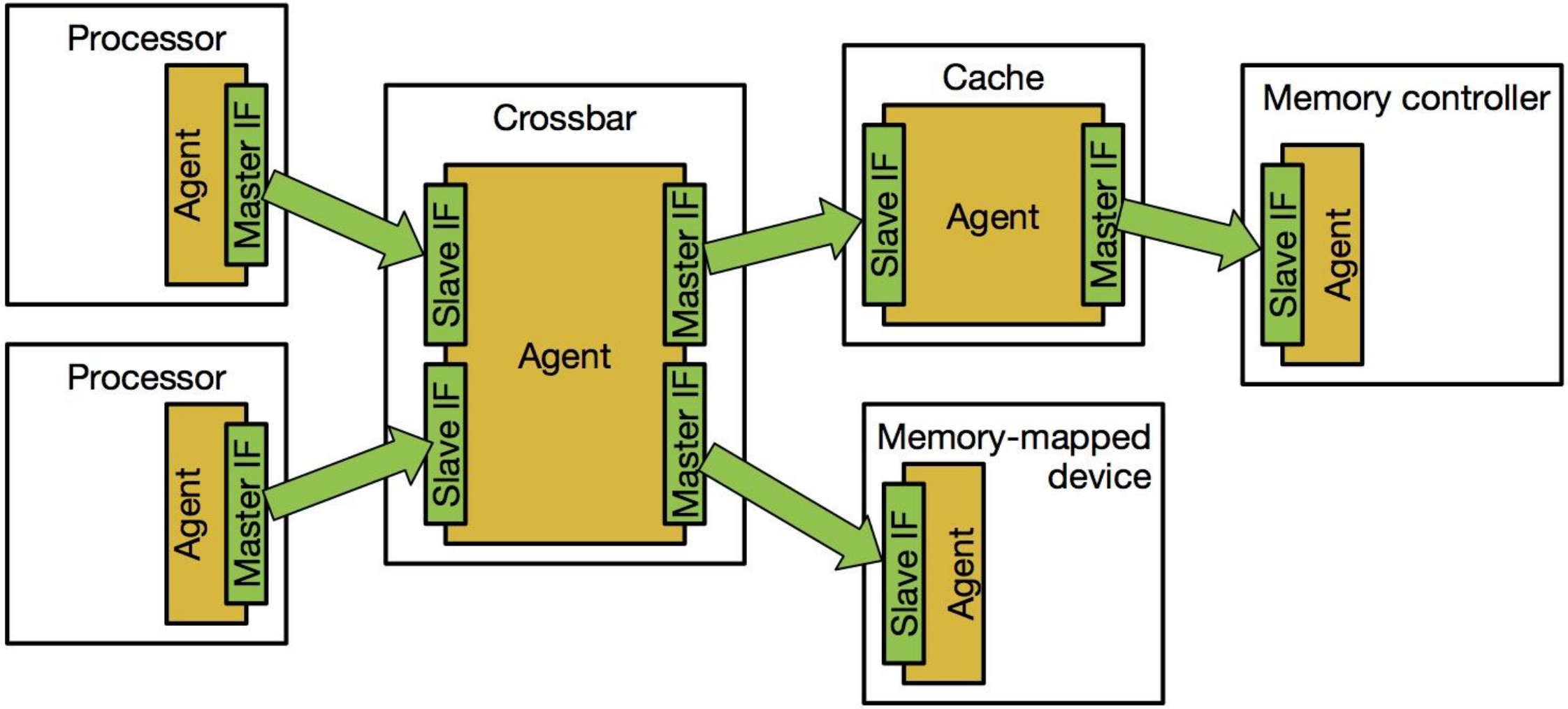
- What happens when negotiation fails?
  - Error at Scala runtime
- What doesn't happen?
  - Chisel elaboration
  - Verilog emission
  - Verilog compilation
  - Simulation execution
  - Waveform debugging





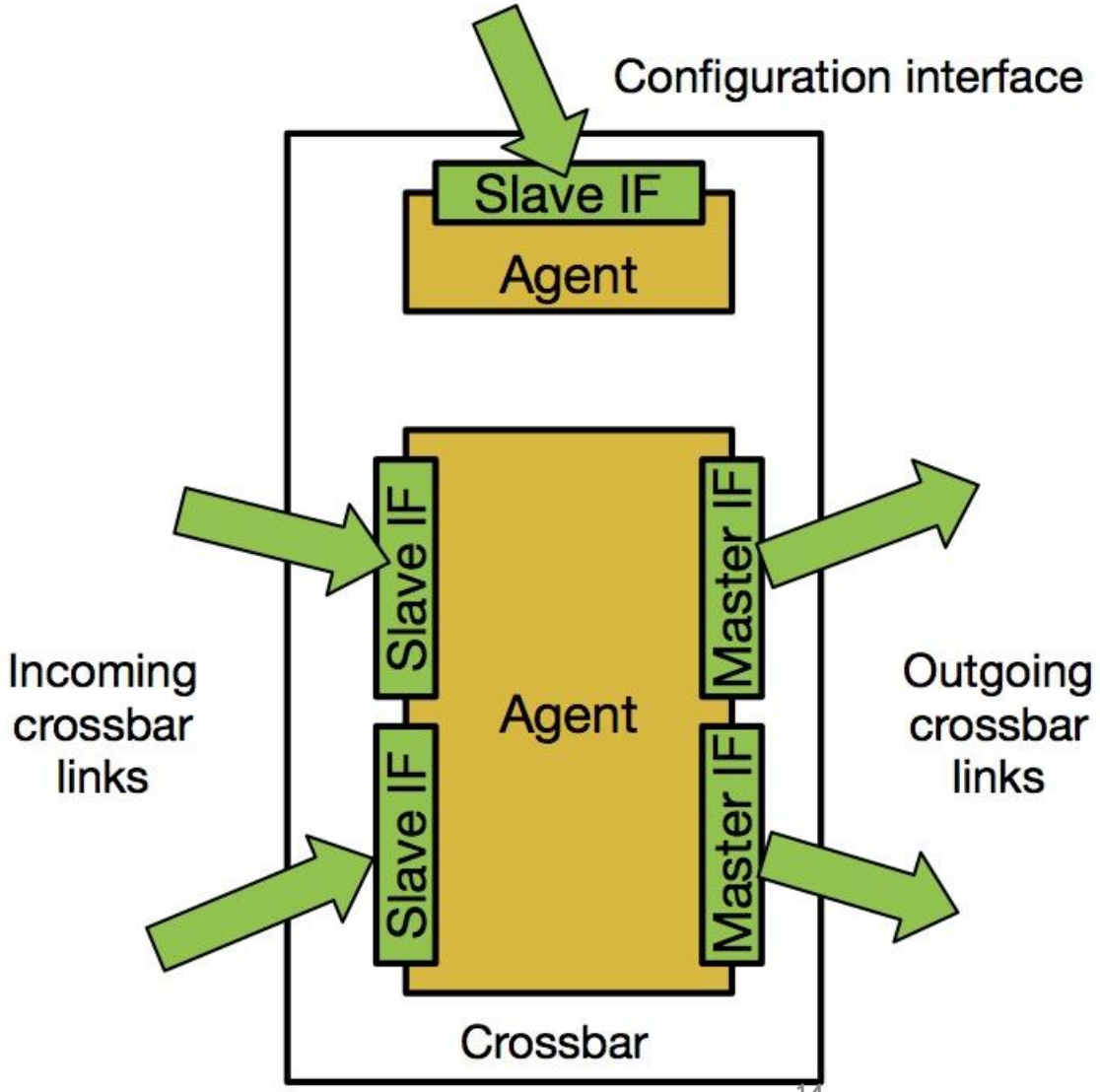
TileLink

TileLink: Graphical Structure

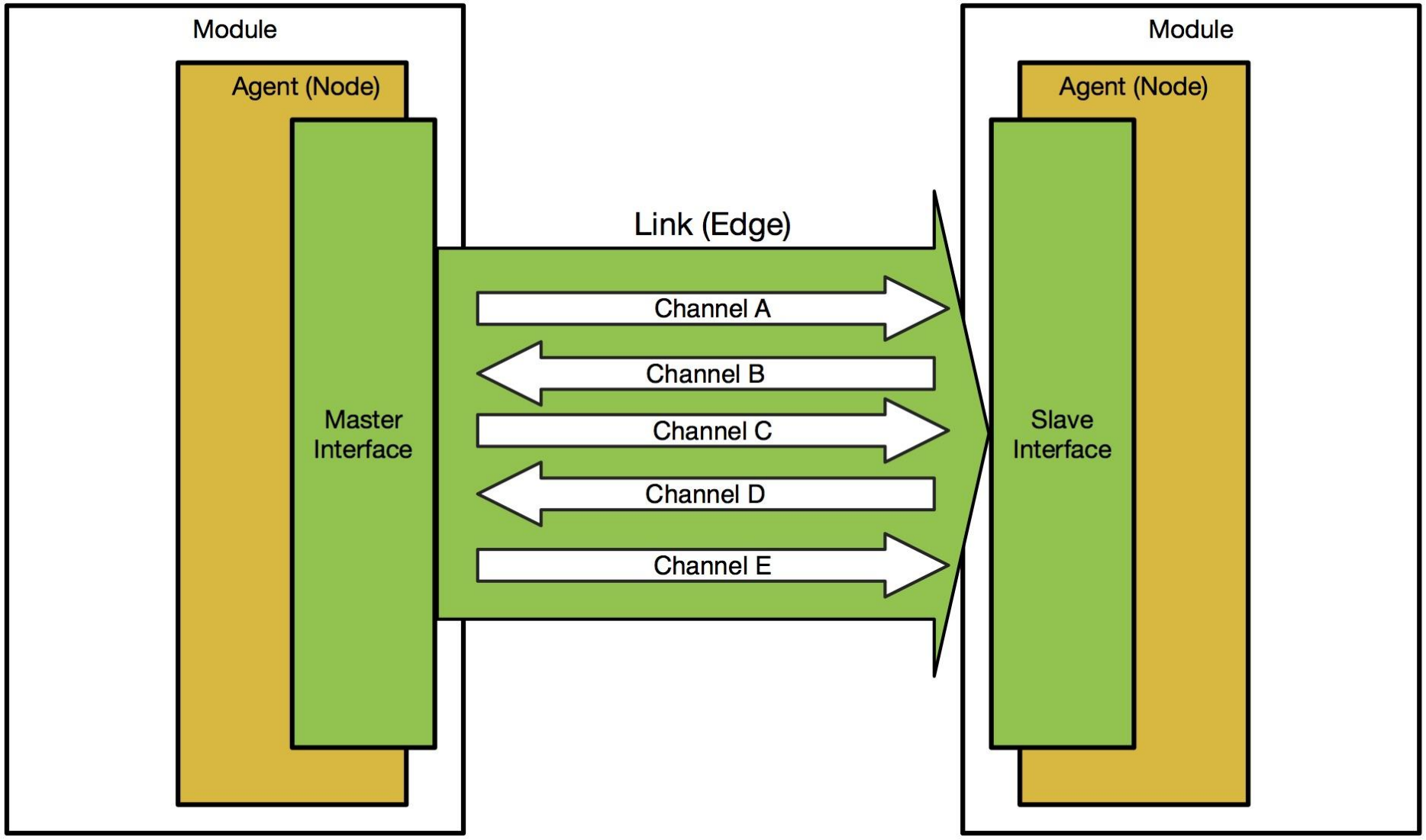


TileLink: Graphical Structure

- Directed Acyclic Graph (DAG) (again)
- Agent (Node)
  - point where messages are created
- Link (Edge)
  - a directed pairing between two agents' master and slave interface
- Modules may have many agents and agents may have many links



TileLink: Directed Channels



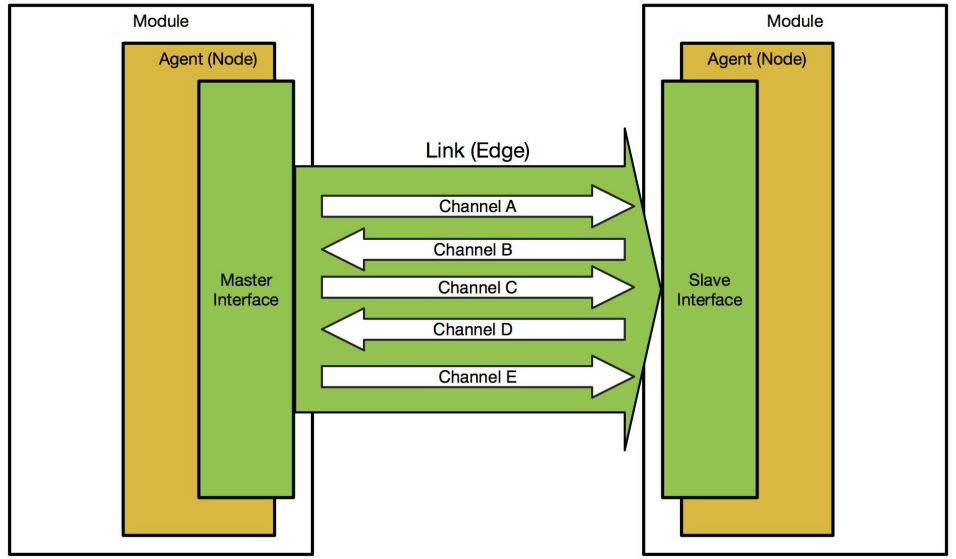
TileLink: Modularity

- Spec defines threes levels of conformance
- Rocket-Chip uses Diplomacy to achieve even finer granularity
- Individual links specialized to the types of messages sent

	TL-UL	TL-UH	TL-C
Cache line transfers	.	.	y
Channels B+C+E	.	.	y
Multibeat operations	.	y	y
Atomic accesses	.	y	y
Hint operations	.	y	y
Get/Put accesses	y	y	y

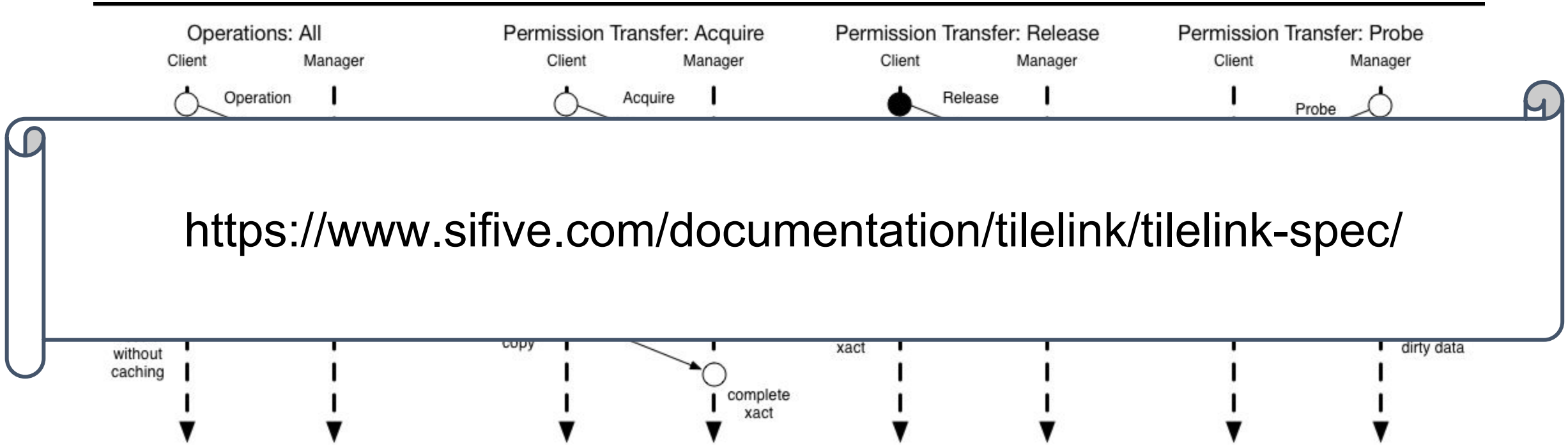
TileLink: Composability

- Generality of interfaces
  - All agents use the same transaction structure
- Scalability rules for hierarchy
  - DAG prevents deadlock
- Strict prioritization of channels
  - Necessary for individual transactions to make forward progress
- Decoupled handshaking rules
  - Limit when and why messages can be rejected

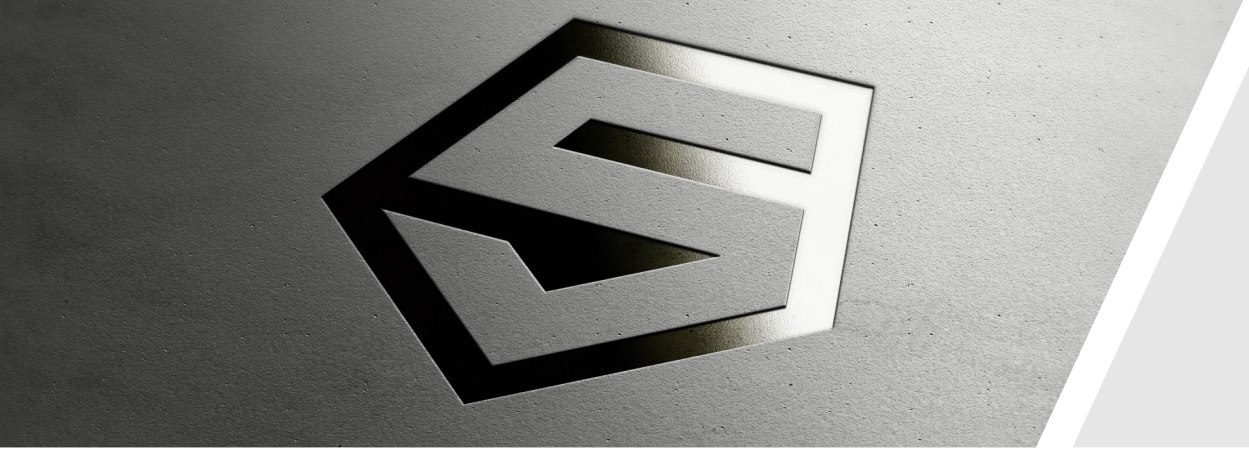


- Deadlock freedom and forward progress guarantees!

TileLink: Transaction Structure



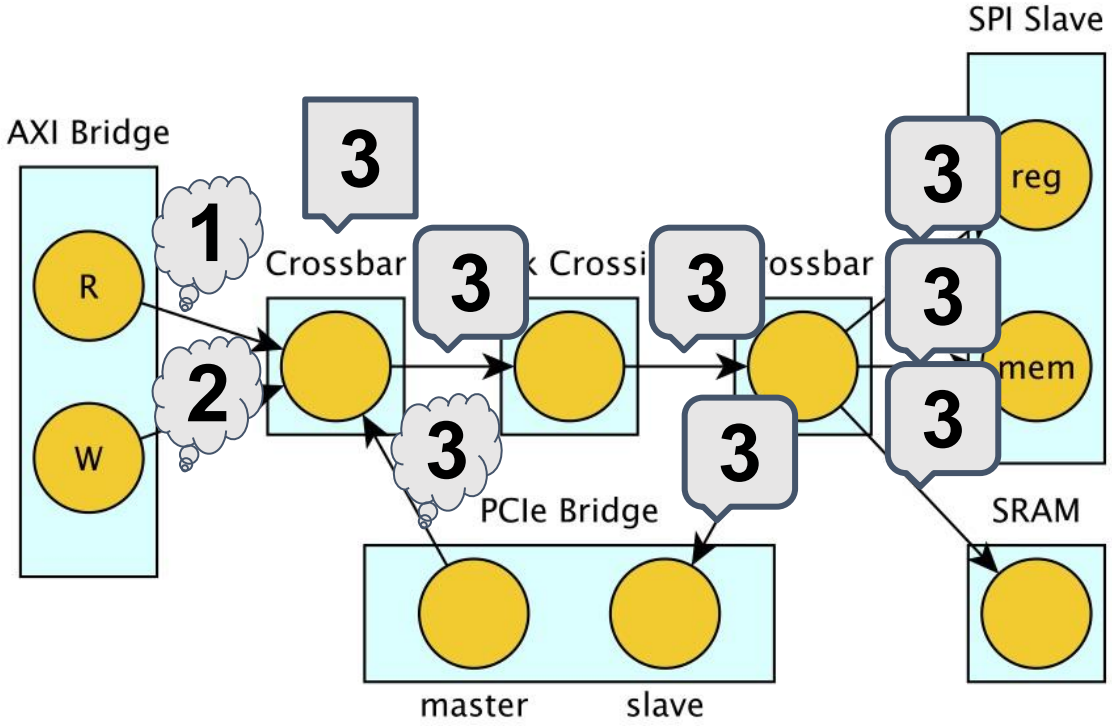
SiFive



Design Patterns

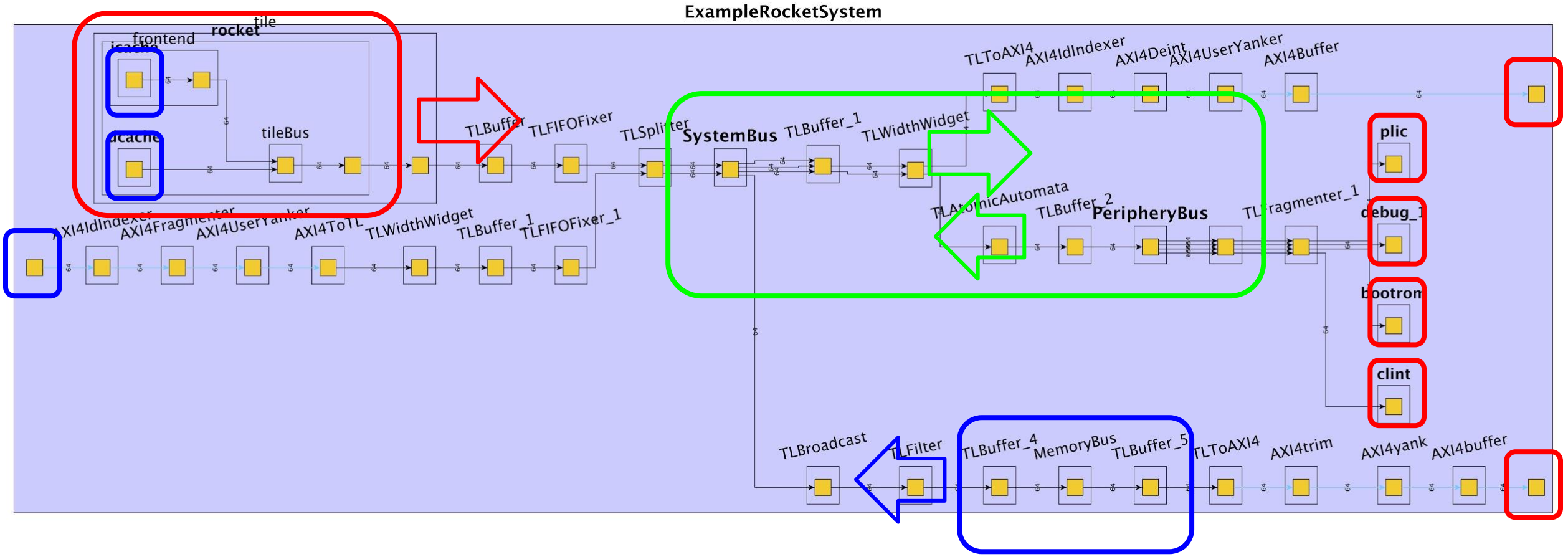
DRYing out parameterization

- Don't Repeat Yourself
  - "Every piece of knowledge must have a single , unambiguous, authoritative representation within a system"



SiFive

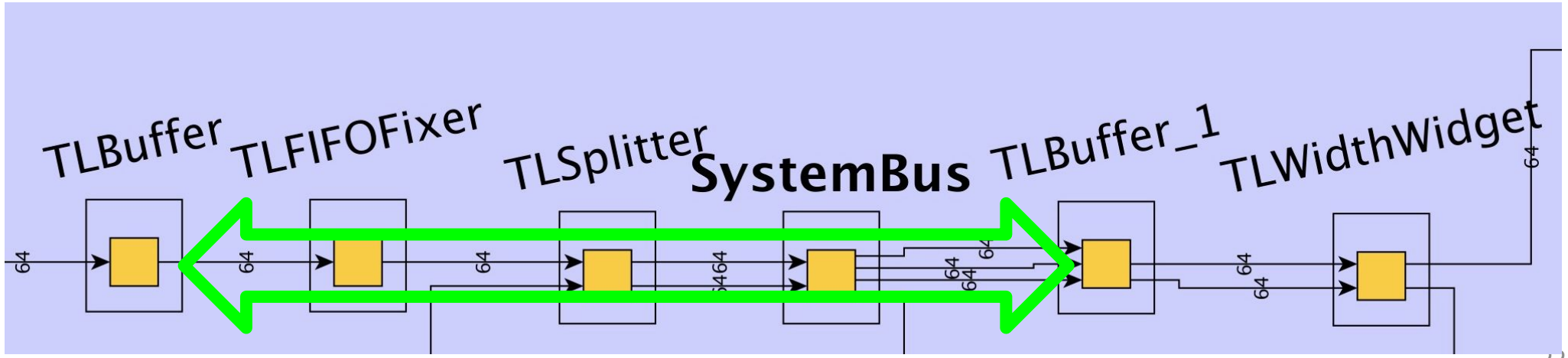
Hardware Generation With A View



SiFive

Combinational Composition

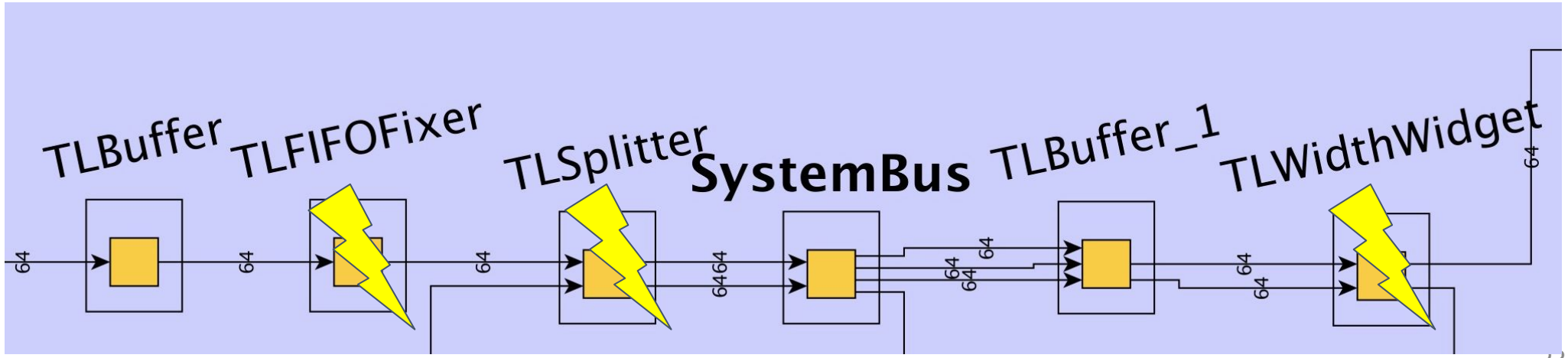
- Thin adapters that each serve a specific, orthogonalized function
- Zero-cycle response time is allowed (support is actually mandatory)
- Simplify verification at no design cost



SiFive

Combinational Composition

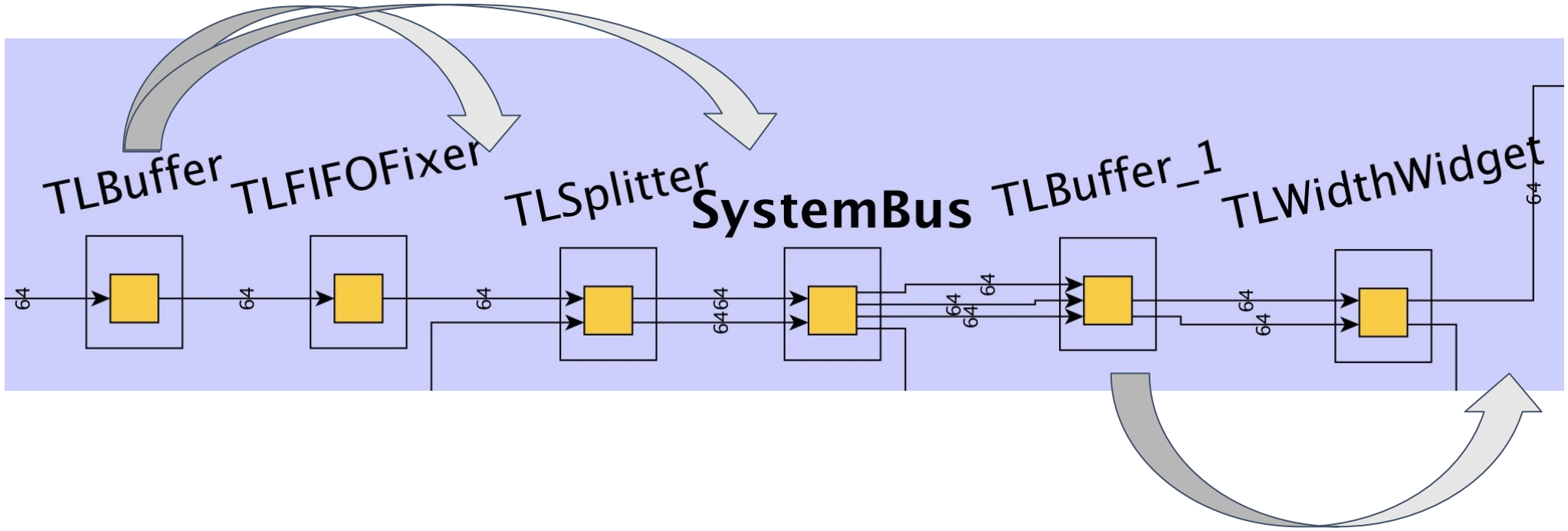
- Modify control signals
  - fragment burst messages into a series of single beat messages
- Modify message field widths
  - widen the width of the data bus plane
- Manage inter-message transaction requirements
  - enforce FIFO ordering across a series of messages and responses



SiFive

Sequential Composition

- Decoupled nature of TileLink interfaces makes it easy to insert buffering at arbitrary point in the graph



Hierarchical Composition

- Any sub-graph can be swapped out for a different sub-graph that provides the same properties
  - Inserting caches (or chains of caches)
  - Filtering addresses into banks
  - Transparently crossing clock domains
  - Transparently converting between protocols
- Scala traits
  - Interfaces that can provide concrete members
  - Support multiple inheritance

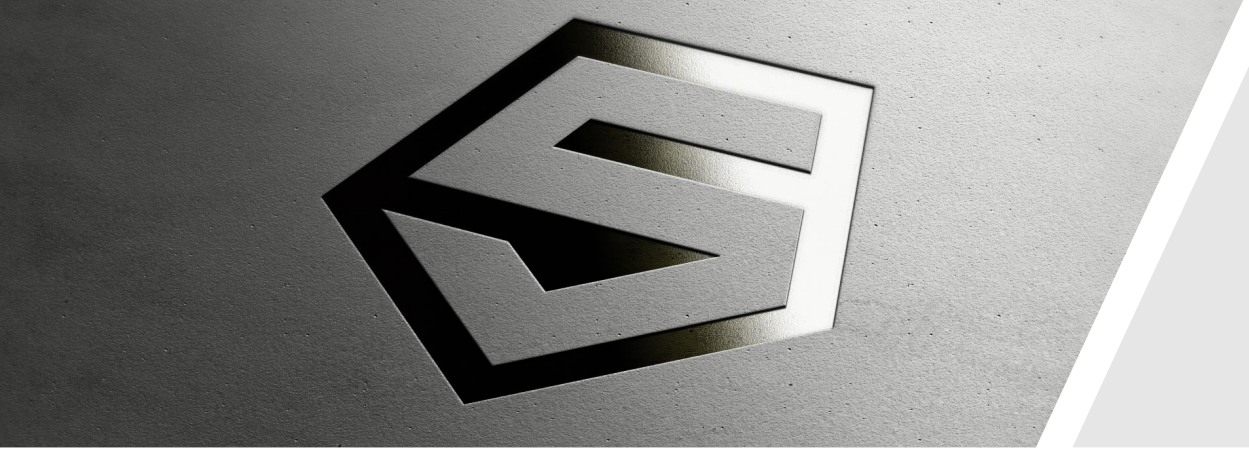
- Build a system from layered components (cake pattern)

Compositional Cake

```
// Compile and elaborate correct hardware:
class SingleCoreSystem extends HasOneCore  with ConnectsIncoherently
class DualCoreSystem   extends HasTwoCores with ConnectsViaBroadcastHub
class DualCoreL2System extends HasTwoCores with ConnectsViaL2Cache

// Fails at Scala compile time due to missing TLSourceNode instance:
class IncompleteSystem extends ConnectsViaL2Cache

// Fails during elaboration due to failed requirement:
class UnsafeSystem     extends HasTwoCores ConnectsIncoherently
  val processorMasterNode = xbar.node
  cores.foreach { c => xbar.node := c.node }
}
```



Deployment

Rocket-Chip: 2017 Rewrite

- Piecewise conversion to Diplomatic TileLink and AMBA
  - using adapters between sub-graphs with different protocol versions
- Started from slaves and moved inward to masters
- Took about 6 months to complete
- Only remaining non-diplomatic Modules are the top-level test harness and leaf modules inside of the Rocket tiles

Future Work: Diplomacy

- Automatic punching of IOs via cross-module edges
  - Ease arbitrary changes to granularity of module hierarchicalization
- Conversion between clock domains
  - Cross at boundary of automatically inserted module wrappers
- Lightweight support for interrupts and other simple types that need to be routed to arbitrary locations, including to top-level IOs
- Diplomatic RoCC

Future Work: TileLink

- Critical Word First for cache blocks
- Richer performance hints
- Cache coherence protocol parameterization
- Deadlock freedom and forward progress proofs, formal models
- ChipLink


SiFive