Choosing Connectors

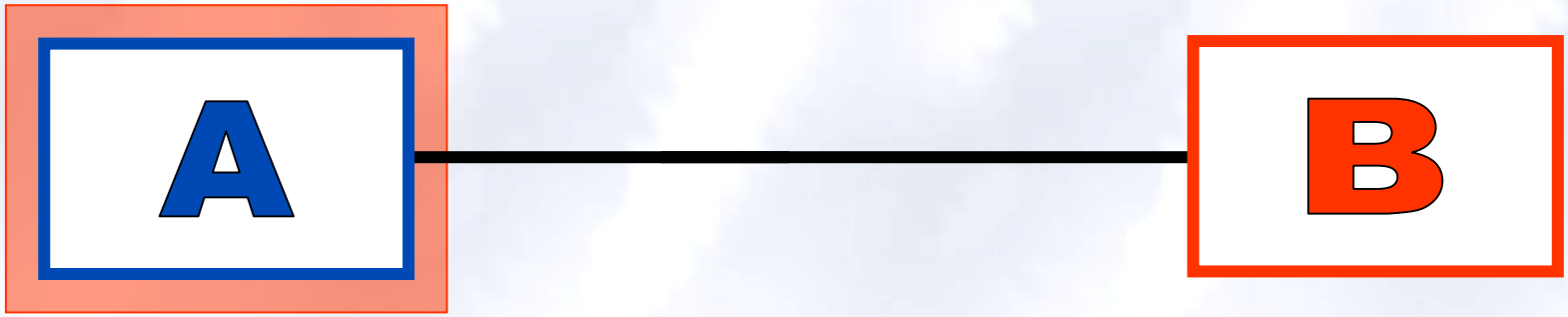
Software Architecture

Role and Challenge of Software Connectors

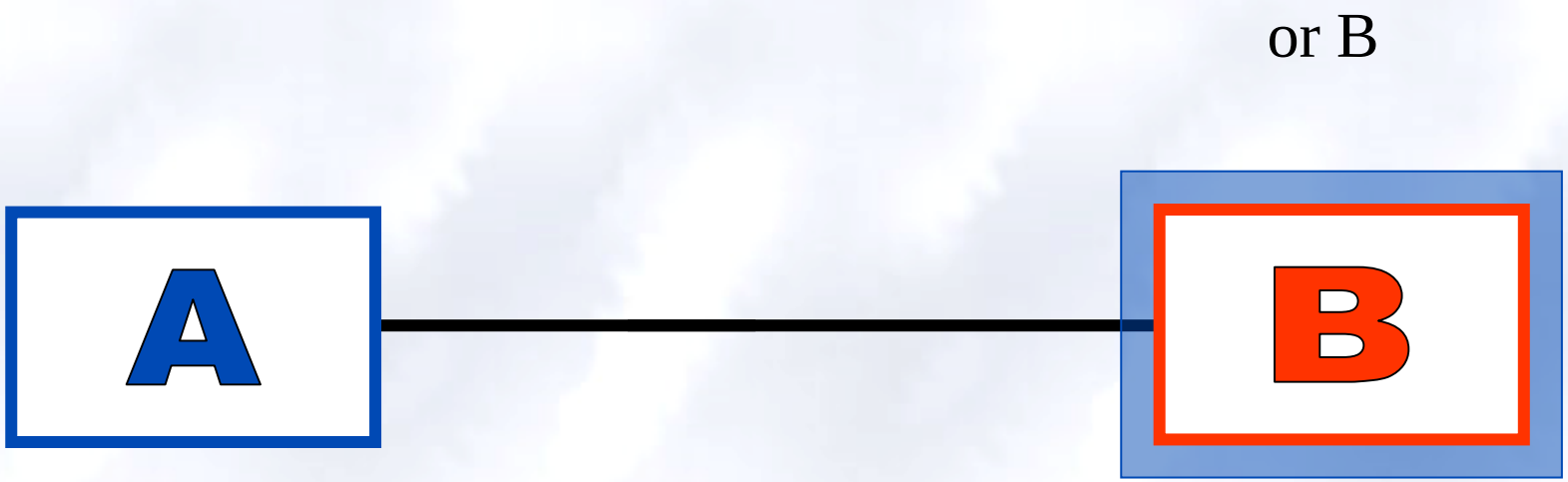
How do we enable components A and B to interact?

Role and Challenge of Software Connectors

Attach adapter to A



Role and Challenge of Software Connectors



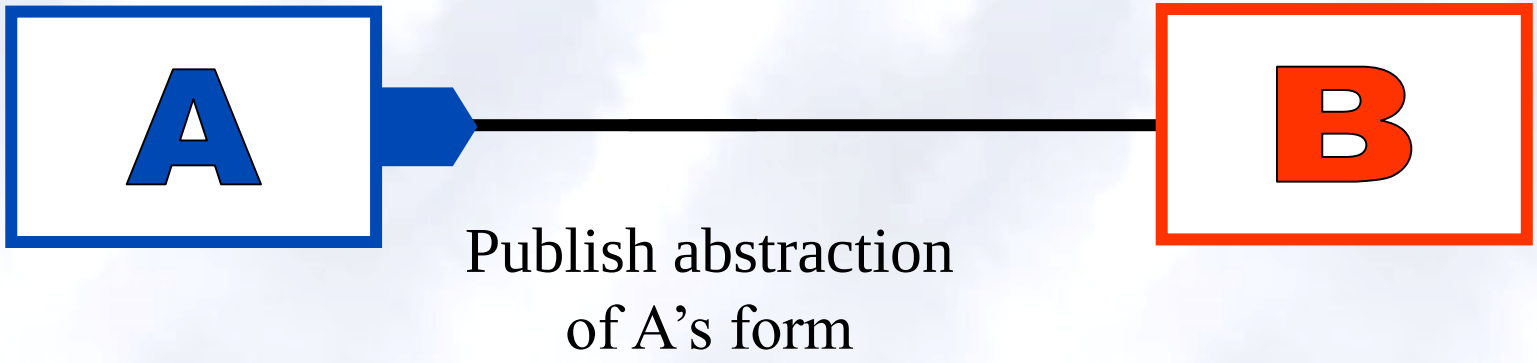
Role and Challenge of Software Connectors

Change A's form to B's form

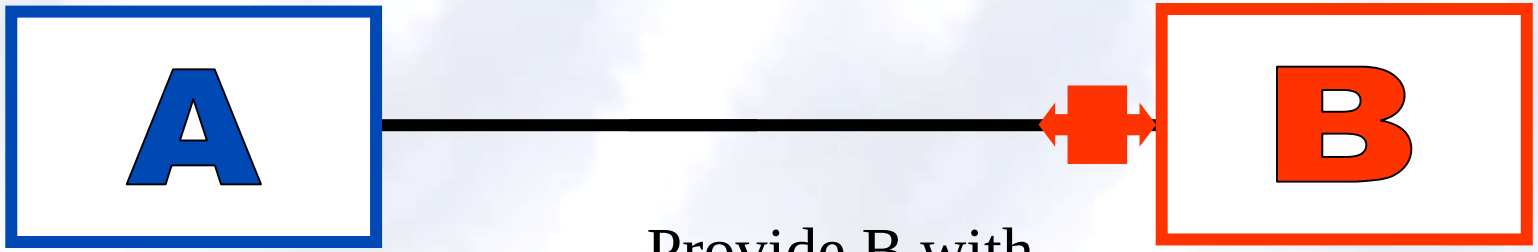
Role and Challenge of Software Connectors

Make B multilingual

Role and Challenge of Software Connectors

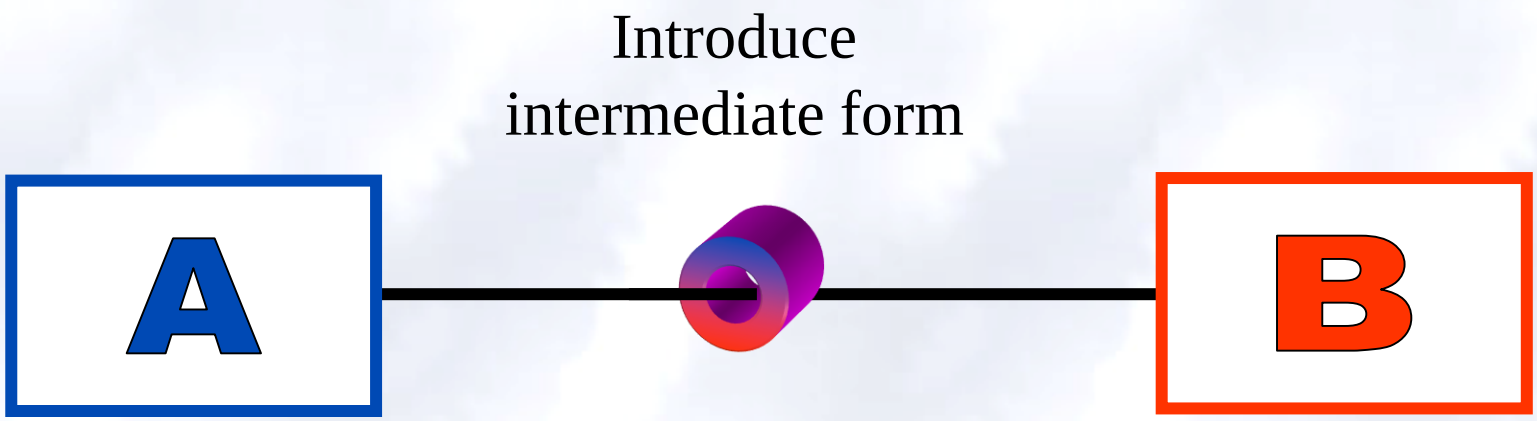


Role and Challenge of Software Connectors

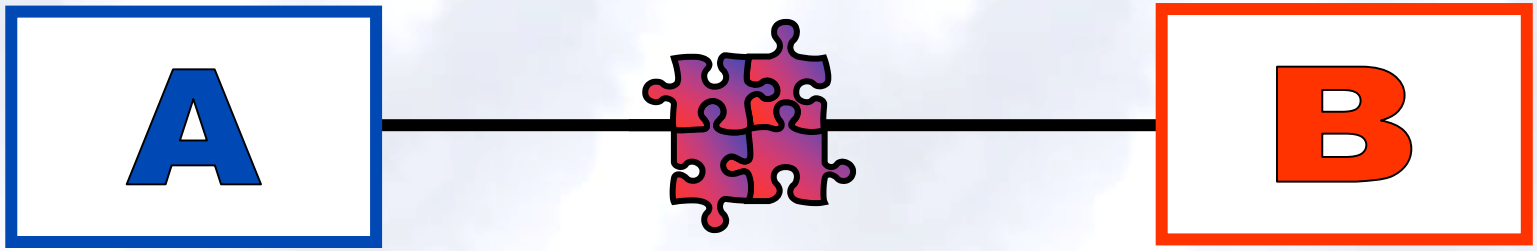


Provide B with
import/export converter

Role and Challenge of Software Connectors

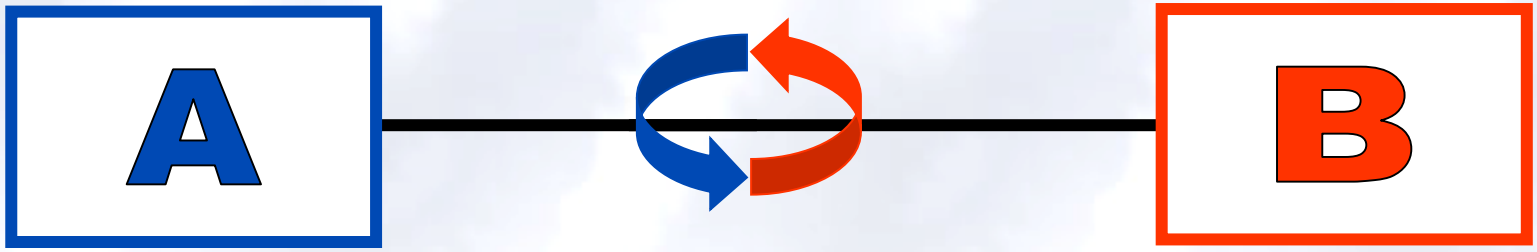


Role and Challenge of Software Connectors



Negotiate to find
common form for A and B

Role and Challenge of Software Connectors



Transform on the fly

Role and Challenge of Software Connectors

Maintain multiple versions of A

Role and Challenge of Software Connectors

Separate B's "essence" from its packaging

Role and Challenge of Software Connectors

What is the right answer?

How Does One Select a Connector?

- Determine a system's interconnection and interaction needs
  - Software interconnection models can help
- Determine roles to be fulfilled by the system's connectors
  - Communication, coordination, conversion, facilitation
- For each connector
  - Determine its appropriate type(s)
  - Determine its dimensions of interest
  - Select appropriate values for each dimension
- For multi-type, i.e., composite connectors
  - Determine the atomic connector compatibilities
  - a trade-off analysis among multiple possible solutions

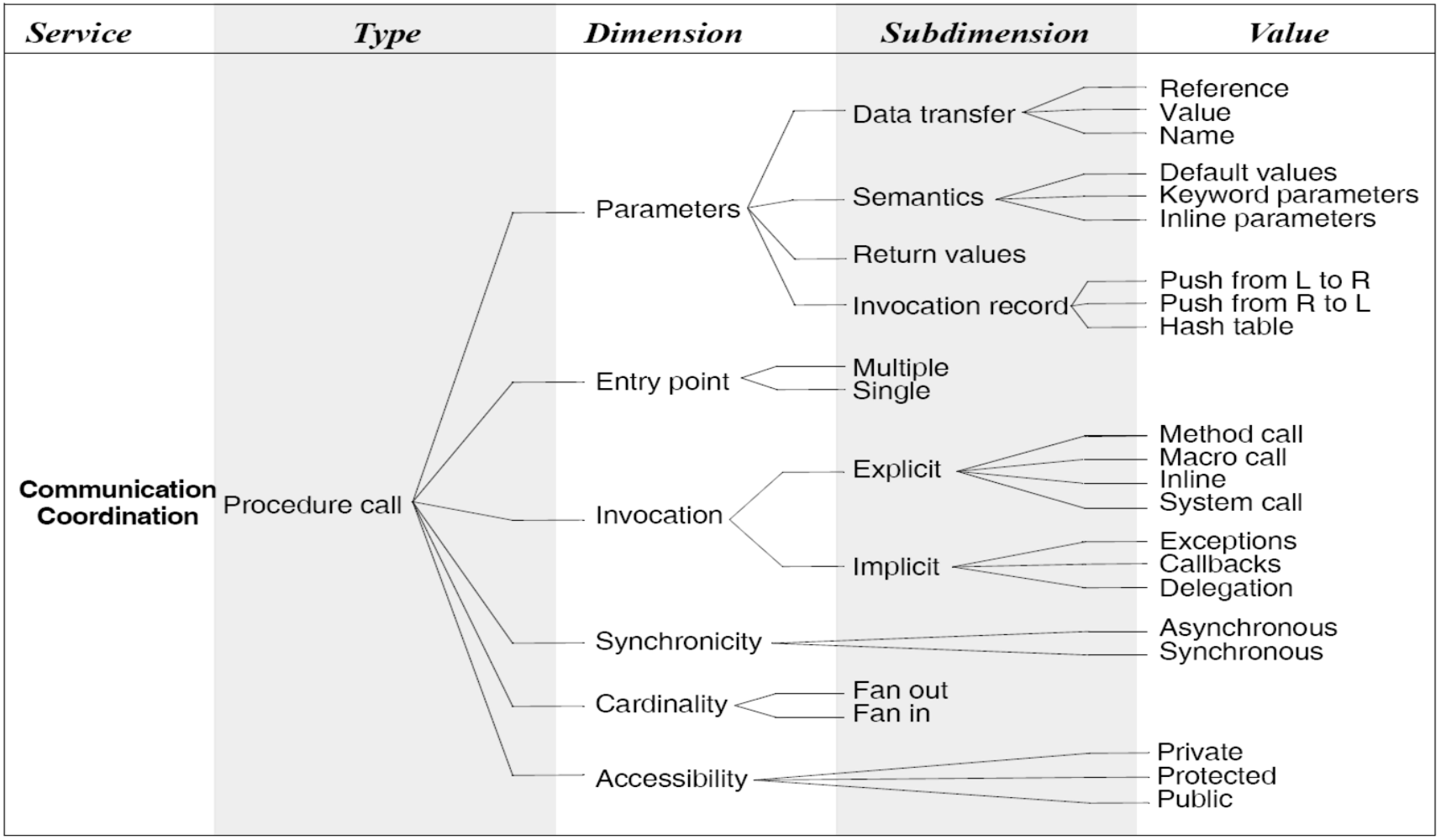
Simple Example

- System components will execute in two processes on the same host
  - Mostly intra-process
  - Occasionally inter-process
- The interaction among the components is synchronous
- The components are primarily computation-intensive
  - There are some data storage needs, but those are secondary

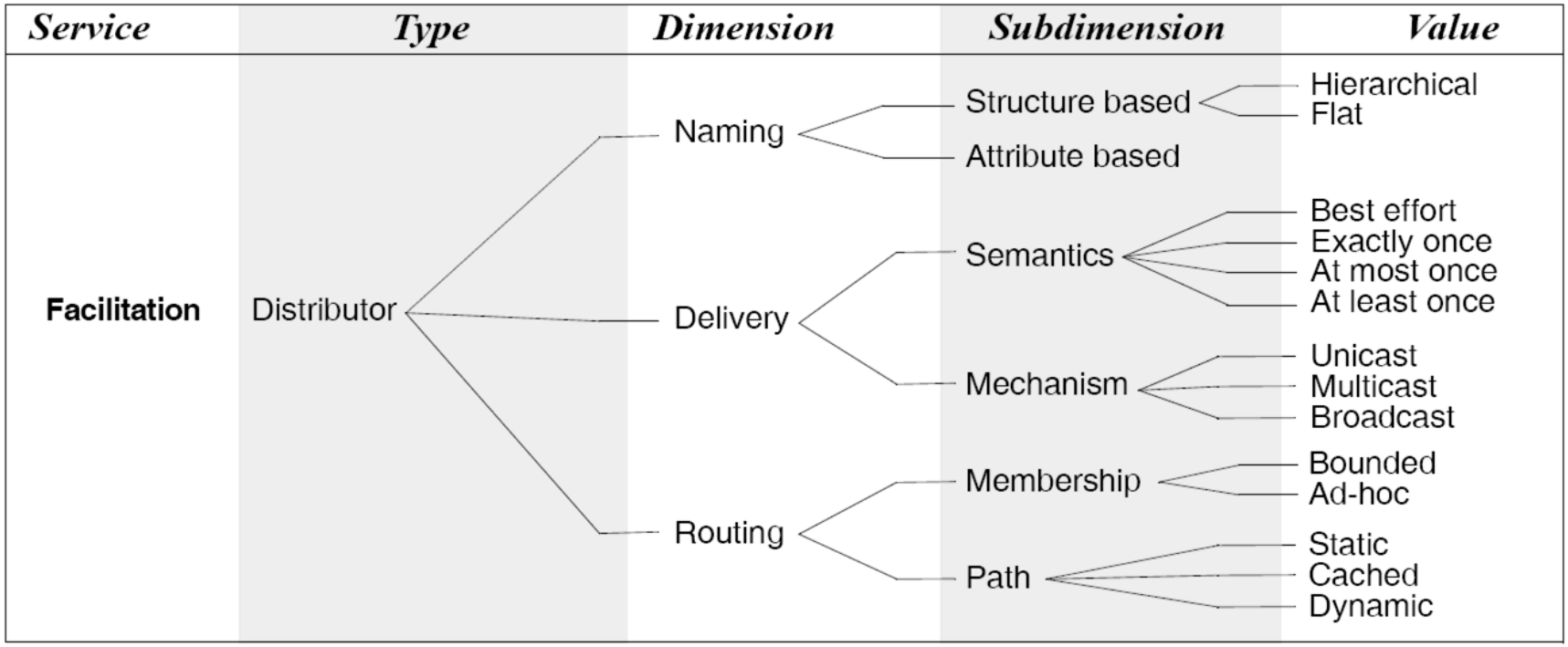
Simple Example (cont'd)

- Select procedure call connectors for intra-process interaction
- Combine procedure call connectors with distributor connectors for inter-process interaction
  - RPC
- Select the values for the different connector dimensions
  - What are the appropriate values?
  - What values are imposed by your favorite programming language(s)?

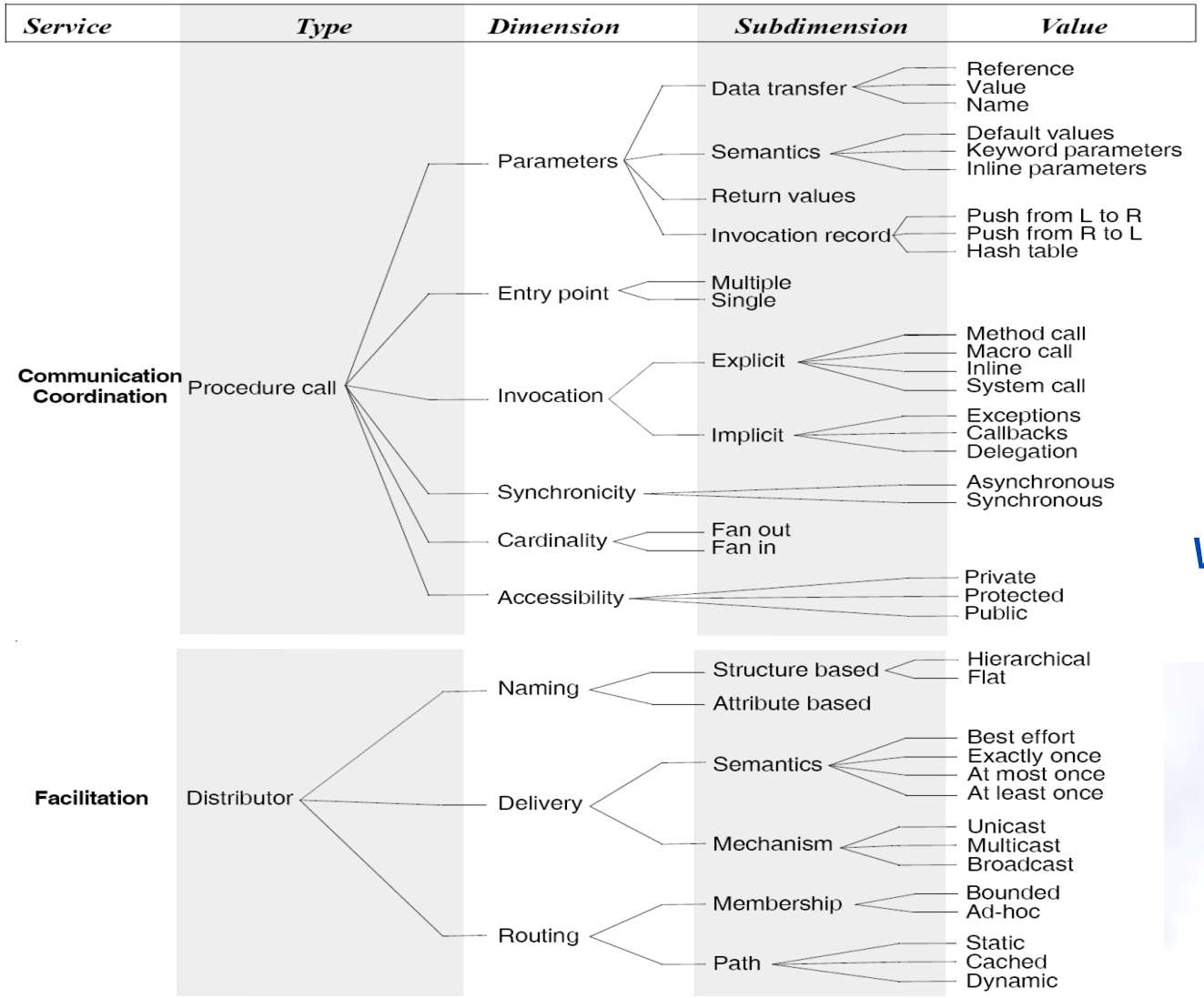
Procedure Call Connectors Revisited



Distributor Connectors Revisited



Two Connector Types in Tandem



Select the appropriate values for PC and RPC!

Software Interconnection Models

- Interconnection models (IM) as defined by Perry
  - Unit interconnection
  - Syntactic interconnection
  - Semantic interconnection
- All three are present in each system
- Are all equally appropriate at architectural level?

Unit Interconnection

- Defines relations between system's units
  - Units are components (modules or files)
  - Basic unit relationship is dependency
    - Unit-*IM* = ({units},{"depends on"})
- Examples
  - Determining context of compilation
    - e.g., C preprocessor
    - *IM* = ({files},{"include"})
  - Determining recompilation strategies
    - e.g., Make facility
    - *IM* = ({compile_units},{"depends on","has changed"})
  - System modeling
    - e.g., RCS, DVS, SVS, SCCS
    - *IM* = ({systems, files},{"is composed of"})

Unit Interconnection Characteristics

- Coarse-grain interconnections
  - At level of entire components
- Interconnections are static
- Does not describe component interactions
  - Focus is exclusively on dependencies

Syntactic Interconnection

- Describes relations among syntactic elements of programming languages
  - Variable definition/use
  - Method definition/invocation
    - *IM* = ({*methods*, *types*, *variables*, *locations*}, {"*is def at*", "*is set at*", "*is used at*", "*is del from*", "*is changed to*", "*is added to*"})
- Examples
  - Automated software change management
    - e.g., Interlisp's masterscope
  - Static analysis
    - e.g., Detection of unreachable code by compilers
  - Smart recompilation
    - Changes inside unit ➔ recompilation of only the changes
  - System modeling
    - Finer level of granularity than unit-IM

Syntactic Interconnection Characteristics

- Finer-grain interconnections
  - At level of individual syntactic objects
- Interconnections are static & dynamic
- Incomplete interconnection specification
  - Valid syntactic interconnections may not be allowed by semantics
  - Operation ordering, communication transactions
    - e.g., Pop on an empty stack
  - Violation of (intended) operation semantics
    - e.g., Trying to use calendar ***add*** operation to add integers

Semantic Interconnection

- Expresses how system components are meant to be used

  Component designers' intentions
- Captures how system components are actually used

  Component users' (i.e., system builders') intention
- Interconnection semantics can be formally specified

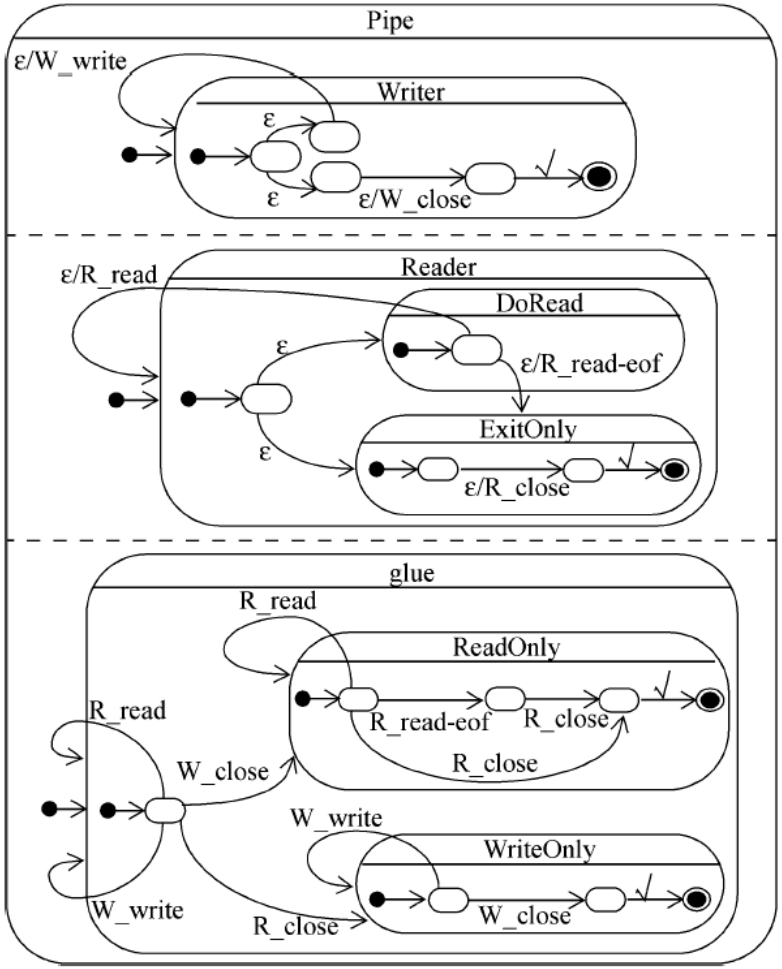
  Pre- & post-conditions

  Dynamic interaction protocols (e.g. CSP, FSM)

  - *IM* = ({*methods*, *types*, *variables*, ..., *predicates*}, {"*is set at*", "*is used at*", "*calls*", "*called by*", ..., "*satisfies*"})

Example of Semantic Interconnection

```
connector Pipe =
  role Writer = write → Writer □ close → ✓
  role Reader =
    let ExitOnly = close → ✓
    in let DoRead = (read → Reader
                    □ read-eof → ExitOnly)
    in DoRead □ ExitOnly
  glue = let ReadOnly = Reader.read → ReadOnly
                    □ Reader.read-eof
                                → Reader.close → ✓
                    □ Reader.close → ✓
     in let WriteOnly = Writer.write → WriteOnly
                     □ Writer.close → ✓
    in Writer.write → glue
     □ Reader.read → glue
     □ Writer.close → ReadOnly
     □ Reader.close → WriteOnly
```



Semantic Interconnection Characteristics

- Builds on syntactic interconnections
- Interconnections are static & dynamic
- Complete interconnection specification
  - Specifies both syntactic & semantic interconnection validity
- Necessary at level of architectures
  - Large components
  - Complex interactions
  - Heterogeneity
  - Component reuse
- What about ensuring other properties of interaction?
  - Robustness, reliability, security, availability, ...

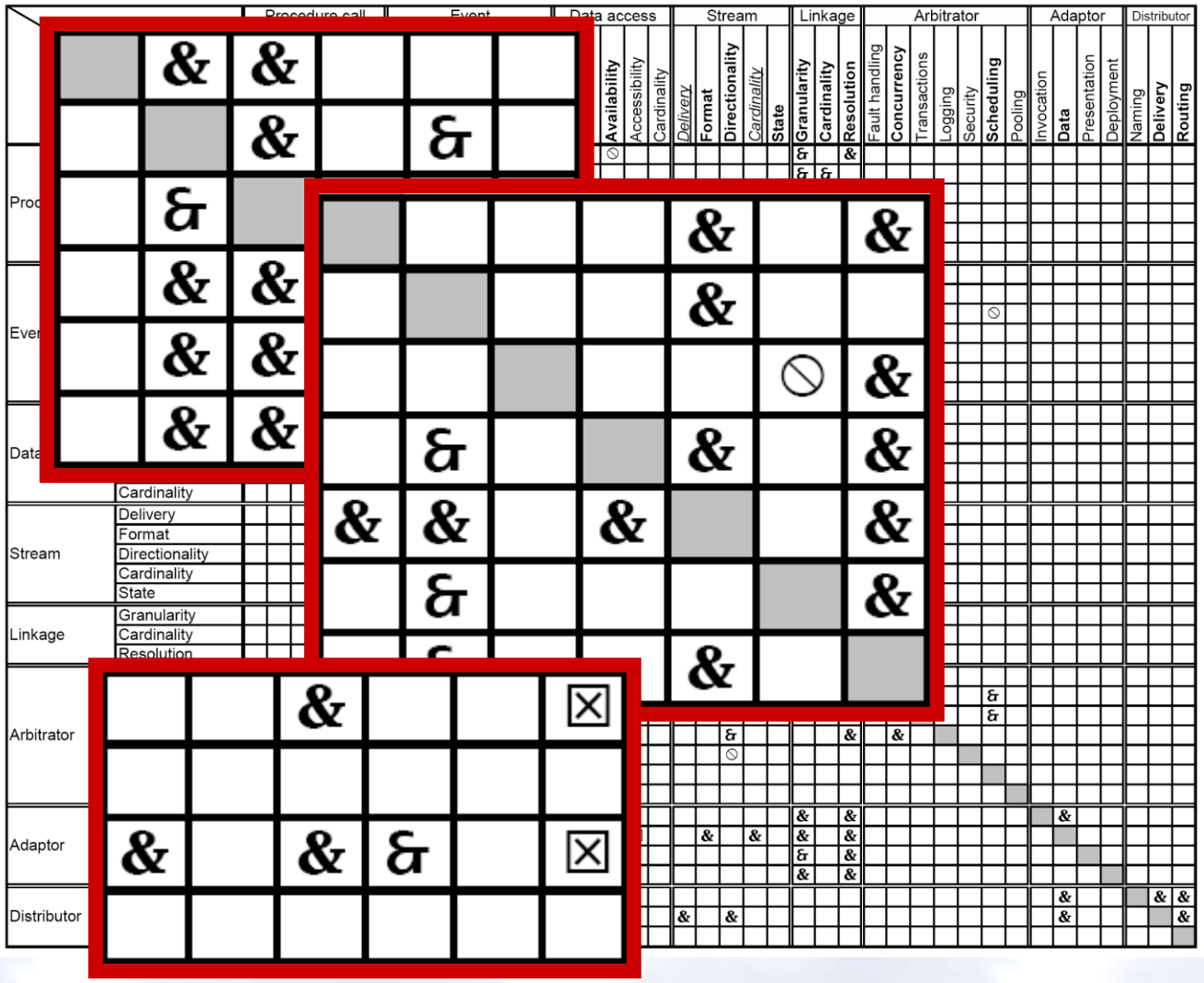
Composing Basic Connectors

- In many systems a connector of multiple types may be required to service (a subset of) the components
- All connectors cannot be composed
  - Some are naturally interoperable
  - Some are incompatible
  - All are likely to require trade-offs
- The composition can be considered at the level of connector type dimensions and subdimensions

Connector Dimension Inter-Relationships

- Requires – 
  Choice of one dimension mandates the choice of another
- Prohibits – 
  Two dimensions can never be composed into a single connector
- Restricts – 
  Dimensions are not always required to be used together
  Certain dimension combinations may be invalid
- Cautions – 
  Combinations may result in unstable or unreliable connectors

Dimension Inter-Relationships in a Nutshell



Well Known Composite Connectors

- Grid connectors (e.g., Globus)
  - Procedure call
  - Data access
  - Stream
  - Distributor
- Peer-to-peer connectors (e.g., Bittorrent)
  - Arbitrator
  - Data access
  - Stream
  - Distributor
- Client-server connectors
- Event-based connectors