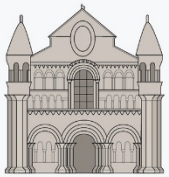
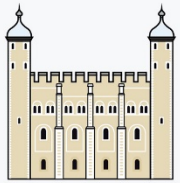
Architectural Styles – Part II

Truong Ho-Quang
truongh@chalmers.se


ROMANESQUE


NORMAN

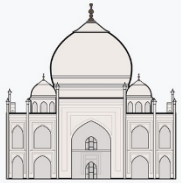

GOTHIC


MEDIEVAL


RENAISSANCE


TUDOR


ELIZABETHAN


INDOISLAMIC


BAROQUE


JACOBEAN

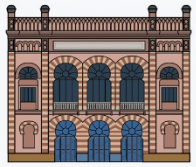

PALLADIAN

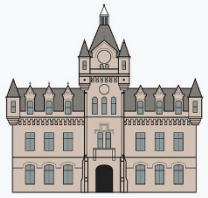

ROCOCO


GEORGIAN


NEOCLASSICAL

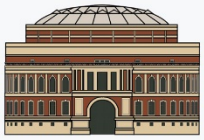

GOTHIC REVIVAL

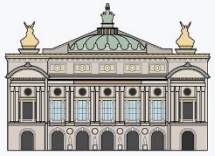

MOORISH REVIVAL

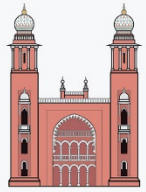

BARONIAL


FEDERAL


REGENCY


ITALIANATE


EMPIRE


INDOSARACENIC


JACOBETHAN


CHICAGO SCHOOL

Schedule

Week		Date	Time	Lecture	Note
36	L1	Wed, 2 Sept	13:15 – 15:00	Introduction & Organization	Ho
37	L2	Wed, 9 Sept	13:15 – 15:00	Architecting Process & Views	Ho
37	S1	Thu, 10 Sept	10:15 – 12:00	<< Supervision/As	
38	L3	Wed, 16 Sept	13:15 - 15:00	Requirements & Quality Attributes	ara
38	S2	Thu, 17 Sept	10:15 – 12:00	<< Supervision/Ass	
38	L4	Fri, 18 Sept	13:15 – 15:00	Architectural Tactics & Roles and Responsib s	Truong Ho
39	S3	Wed, 23 Sept	13:15 – 15:00	<< Supervision/Assign ent>>	TAs
39	L5	Thu, 24 Sept	10:15 – 12:00	Functional Decomposition & Architectural Sty es P1	Truong Ho
39	L6	Fri, 25 Sept	13:15 – 15:00	Architectural Styles P2	Truong Ho
40	S4	Wed, 30 Sept	13:15 – 15:00	<< Supervision/Assignment>>	TAs
40	L7	Thu, 1 Oct	10:15 – 12:00	Architectural Styles P3	Sam Jobara
40	L8	Fri, 2 Oct	13:00 – 15:00	Guest Lecture: Scaling DevOps – GitHub's Journey from 500+ to 1500+ People	Johannes Nicolai
41	S5	Wed, 7 Oct	13:15 – 15:00	<< Supervision/Assignment>>	TAs
41	L9	Thu, 8 Oct	10:15 – 12:00	Current Industrial SW Architecture Issues: Software Architectures of Blockchain with Case Study	Sam Jobara
42	L10	Wed, 14 Oct	13:15 – 15:00	Design Principles	Truong Ho
42	S6	Thu, 15 Oct	10:15 – 12:00	<< Supervision/Assignment>>	TAs
42	L11	Fri, 16 Oct	13:15 – 15:00	Guest Lecture: Architecture changes at Volvo Truck's Application System (TAS)	Anders Magnusson
43	L12	Wed, 21 Oct	13:15 – 15:00	Architecture Evaluation	Truong Ho
43	L13	Thu, 22 Oct	10:15 – 12:00	Reverse Engineering & Correspondence	Truong Ho
43		Fri, 23 Oct	13:00 – 15:00	To be determined (exam practice?)	Teachers
44	Exam	30 Oct	8:30 – 12:30		

We are HERE!

Assignment schedule

Week		Date	Lecture	Assignment 1 – Task 1 (A1T1)	Assignment 1 – Task 2 (A1T2)	Assignment 2 (A2)
36	L1	Wed, 2 Sept	Introduction & Organization			
37	L2	Wed, 9 Sept	Architecting Process & Views	A1T1 released		
37	S1	Thu, 10 Sept	<< Supervision/Assignment>>	Planing A1T1		
38	L3	Wed, 16 Sept	Requirements & Quality Attr.			
38	S2	Thu, 17 Sept	<< Supervision/Assignment>>	Work A1T1		
38	L4	Fri, 18 Sept	Tactics & Roles			
39	S3	Wed, 23 Sept	<< Supervision/Assignment>>	Work A1T1		
39	L5	Thu, 24 Sept	Decomposition & Style P1	Hand-in A1T1		
39	L6	Fri, 25 Sept	Architectural Styles P2		A1T2 released	
40	S4	Wed, 30 Sept	<< Supervision/Assignment>>	Feedback A1T1	Work A1T2	
40	L7	Thu, 1 Oct	Architectural Styles P3			
40	L8	Fri, 2 Oct	Industrial lecture 1			
41	S5	Wed, 7 Oct	<< Supervision/Assignment>>		Work A1T2	A2 released
41	L9	Thu, 8 Oct	Industrial lecture 2			
42	L10	Wed, 14 Oct	Design Principles			
42	S6	Thu, 15 Oct	<< Supervision/Assignment>>		Work A1T2	Work A2
42	L11	Fri, 16 Oct	Industrial lecture 3		Hand-in A1T2	
43	L12	Wed, 21 Oct	Architecture Evaluation		Feedback A1T2	
43	L13	Thu, 22 Oct	Reverse Engineering			Hand-in A2
43		Fri, 23 Oct	Exam practice			Tue, 27 Oct: Feedback A2
44	Exam	30 Oct				

Assignment schedule

Week		Date	Lecture	Assignment 1 – Task 1 (A1T1)	Assignment 1 – Task 2 (A1T2)	Assignment 2 (A2)
36	L1	Wed, 2 Sept	Introduction & Organization			
37	L2	Wed, 9 Sept	Architecting Process & Views	A1T1 released		
37	S1	Thu, 10 Sept	<< Supervision/Assignment>>	Planing A1T1		
38	L3	Wed, 16 Sept	Requirements & Quality Attr.			
38	S2	Thu, 17 Sept	<< Supervision/Assignment>>	Work A1T1		
38	L4	Fri, 18 Sept	Tactics & Roles			
39	S3	Wed, 23 Sept	<< Supervision/Assignment>>	Work A1T1		
39	L5	Thu, 24 Sept	Decomposition & Style P1	Hand-in A1T1		
39	L6	Fri, 25 Sept	Architectural Styles P2		A1T2 released	
40	S4	Wed, 30 Sept	<< Supervision/Assignment>>	Feedback A1T1	Work A1T2	
40	L7	Thu, 1 Oct	Architectural Styles P3			
40	L8	Fri, 2 Oct	Industrial lecture 1			
41	S5	Wed, 7 Oct	<< Supervisi		ork A1T2	A2 released
41	L9	Thu, 8 Oct	Industrial lecture			
42	L10	Wed, 14 Oct	Design Principle			
42	S6	Thu, 15 Oct	<< Supervisi		ork A1T2	Work A2
42	L11	Fri, 16 Oct	Industrial lecture		d-in A1T2	
43	L12	Wed, 21 Oct	Architecture Evaluation		Feedback A1T2	
43	L13	Thu, 22 Oct	Reverse Engineering			Hand-in A2
43		Fri, 23 Oct	Exam practice			Tue, 27 Oct: Feedback A2
44	Exam	30 Oct				

Task 2 is NOW released!

Task 2 of Assignment 1

General advices:

- Start early
- Work together
  - Help your team members
  - Make it fun
- Be consistent
  - With your design
  - With the identified architecture drivers

Outline of Topics for Today's Lecture

- Architectural Styles
  - Pipe and Filter
  - Publish-Subscribe
  - (Blackboard)

CONTENTS

Lecture 5

TODAY!

Lecture 7

Architectural style

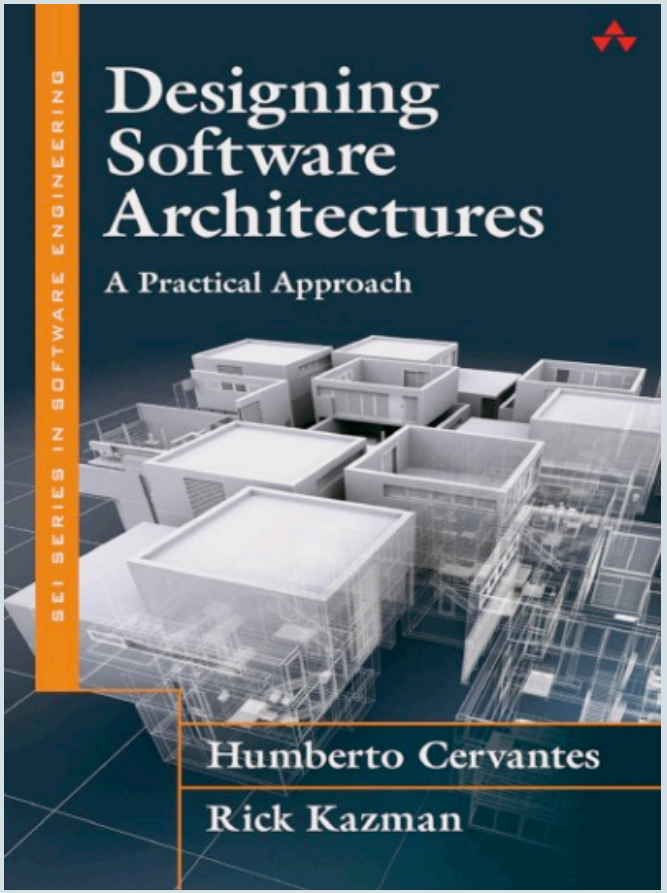
An *architectural style* is defined by:

a set of rules, principles and constraints that prescribe

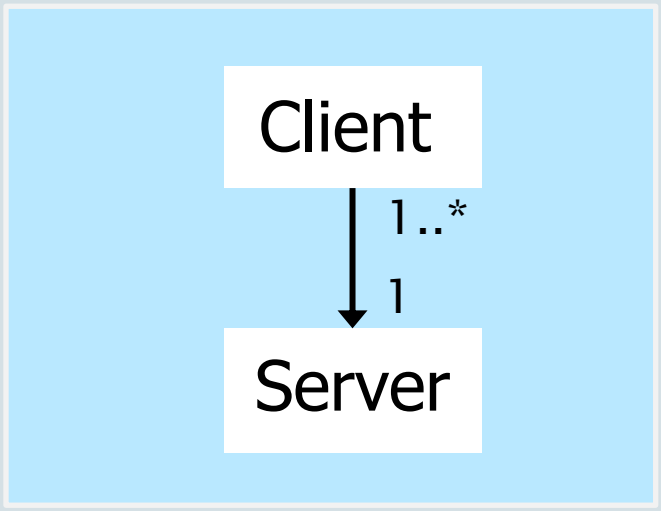
- **vocabulary/metaphor**: which types of components, interfaces & connectors must/may be used in a system.
  Possibly introducing domain-specific types
- **structure**: how components and connectors may be combined
- **behaviour**: how the system behaves
- **guidelines**: these support the application of the style (how to achieve certain system properties)

Deployment patterns for Client-Server

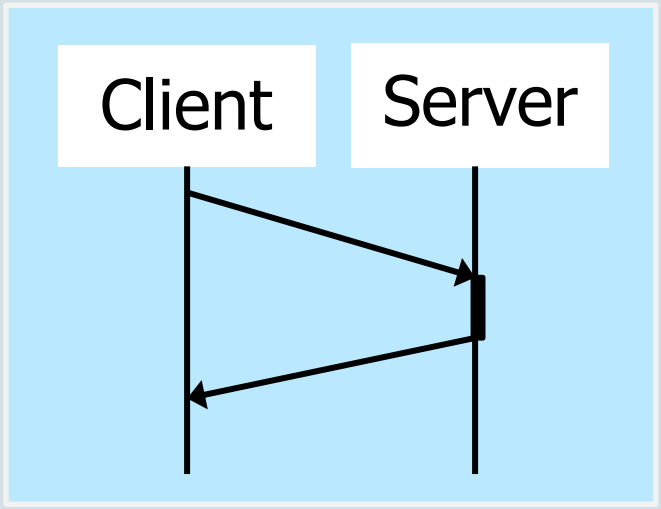
Used diagrams/slides from this book

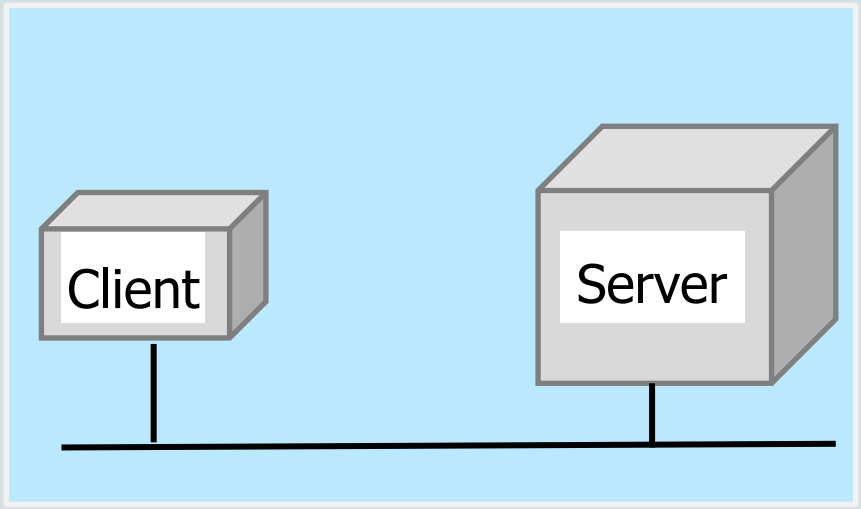


Client–Server (S+B+D)



There is more than one structure to a style!





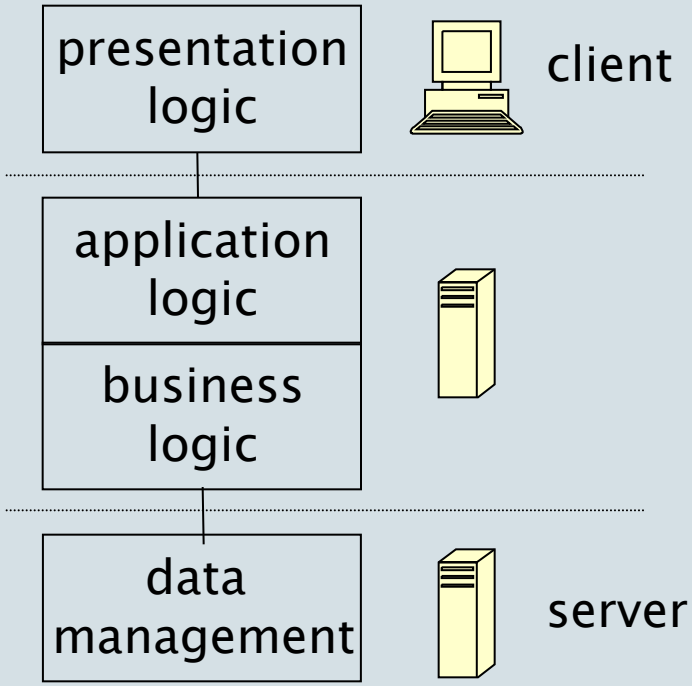
Client / Server Style

Concept: Separation of application in units of change

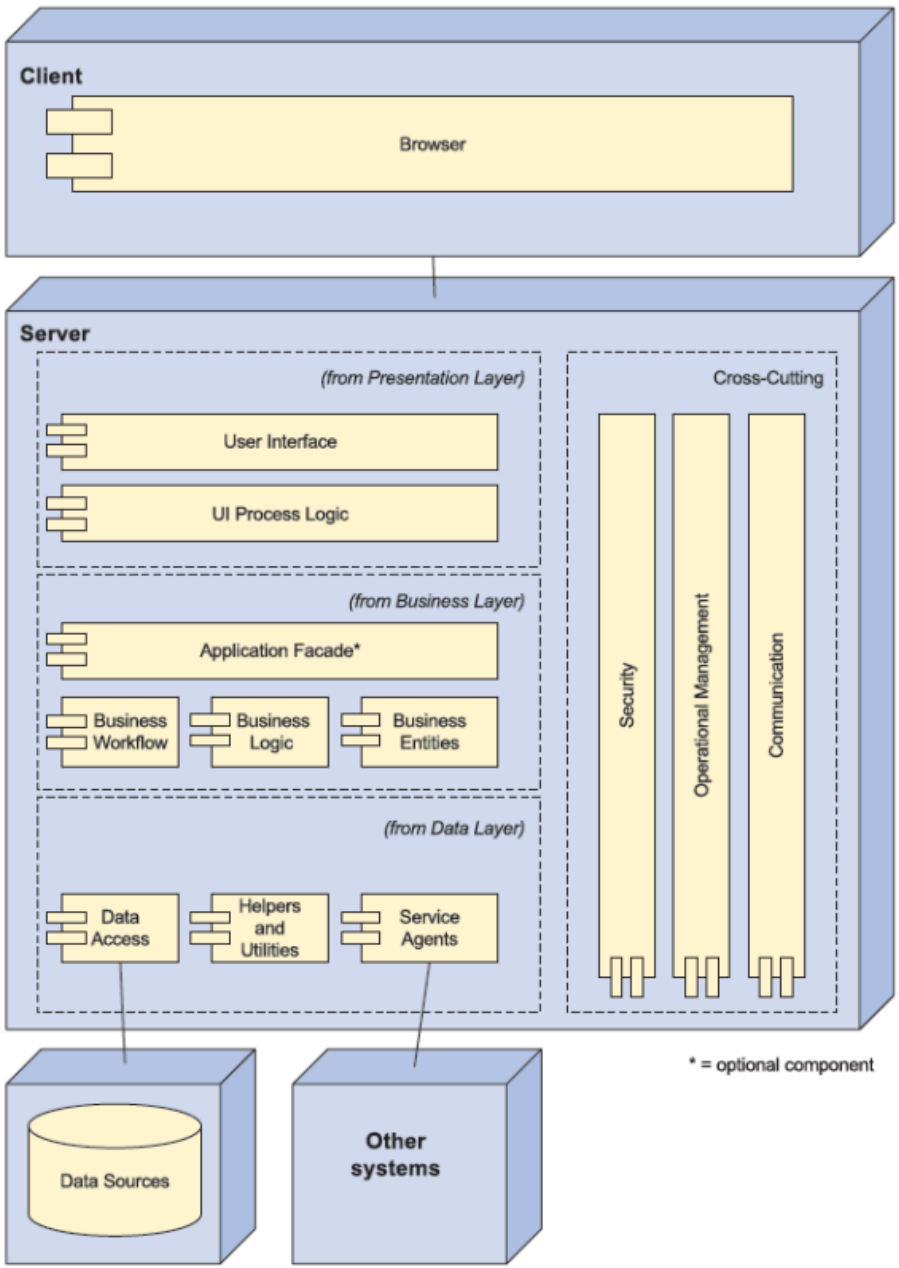
Components: presentation,
application logic,
business logic,
data management

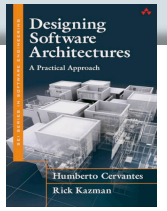
Connector: 'uses' lower layer

Interaction style: request/response




Designing Software Architectures
A Practical Approach
Humberto Cervantes
Rick Kazman


Client
Browser
Server
(from Presentation Layer)
User Interface
UI Process Logic
(from Business Layer)
Application Facade*
Business Workflow
Business Logic
Business Entities
(from Data Layer)
Data Access
Helpers and Utilities
Service Agents
Cross-Cutting
Security
Operational Management
Communication
* = optional component
Data Sources
Other systems



Deployment Patterns

- They provide models on how to physically structure the system to deploy it.

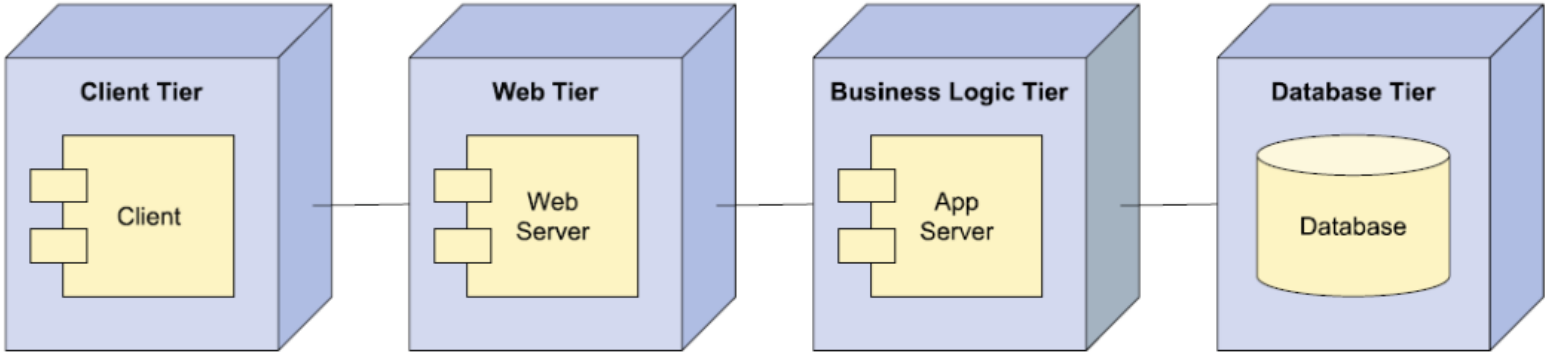


FIGURE 2.6 Four-tier deployment pattern from the *Microsoft Application Architecture Guide* (Key: UML)

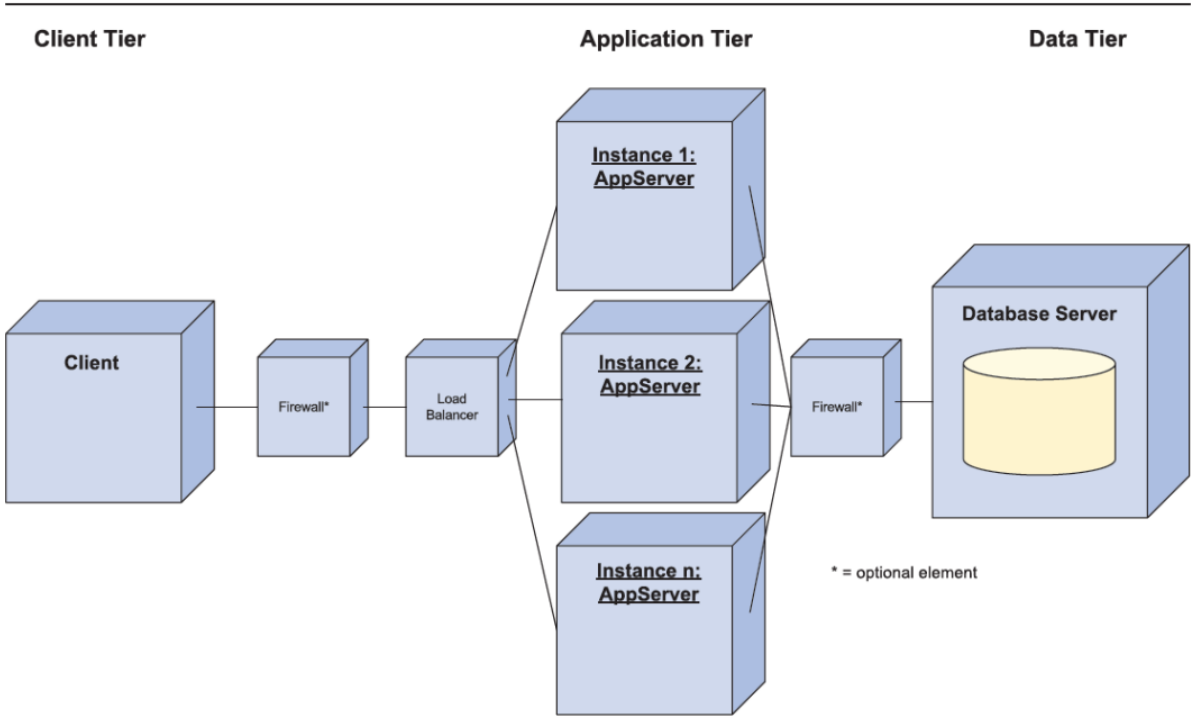
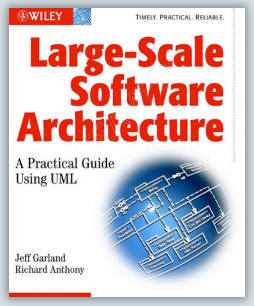


FIGURE 2.7 Load-Balanced Cluster deployment pattern for performance from the *Microsoft Application Architecture Guide* (Key: UML)



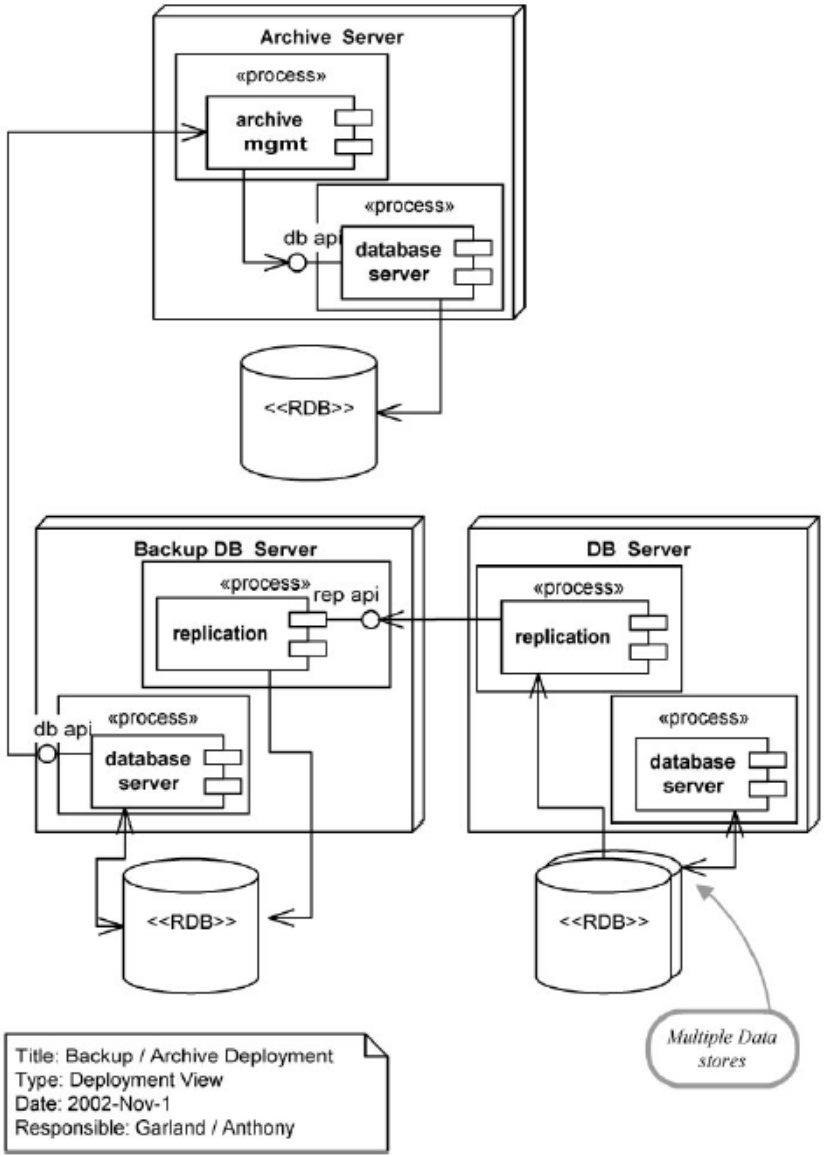
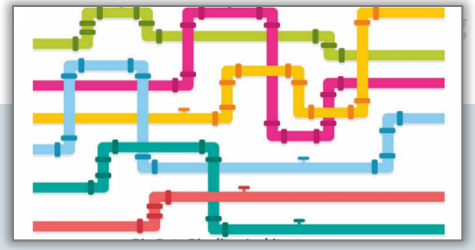


Figure 10.13 Backup/Archive Deployment View

Recap on Styles

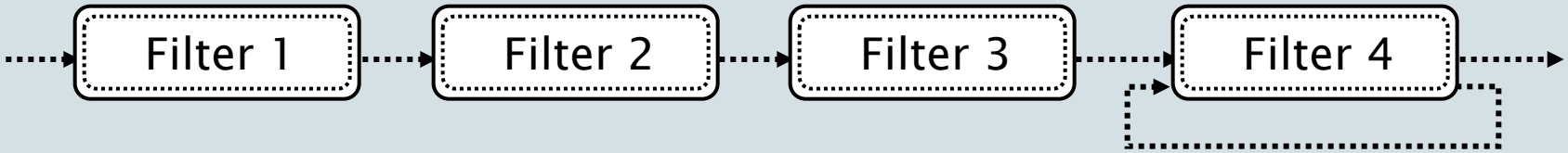
- Conceptual Integrity
- Introduction of Architecture Styles
- Client-Server

CONTENTS



Pipe and Filter Style (1)

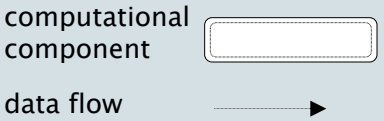
Concept: Series of filters / transformation
where each component is consumer and producer

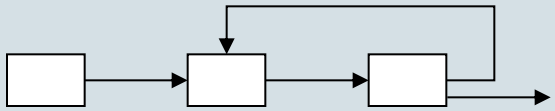


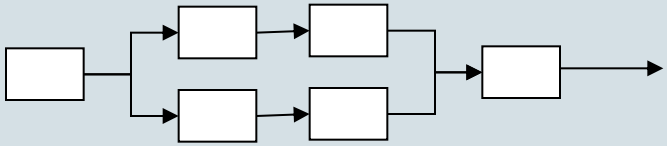
Components: filters / transformations
possibly also: sources and sinks

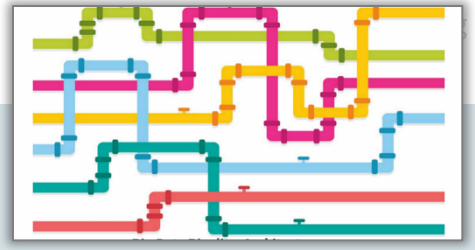
Connectors: pipes;
interaction style: streaming of data

Topology: linear; possible variations:
feedback-loops, splitting pipes





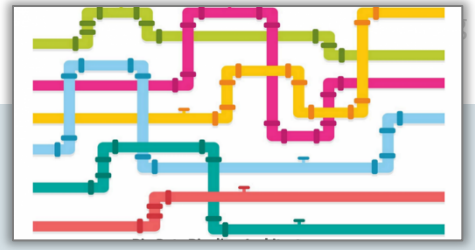




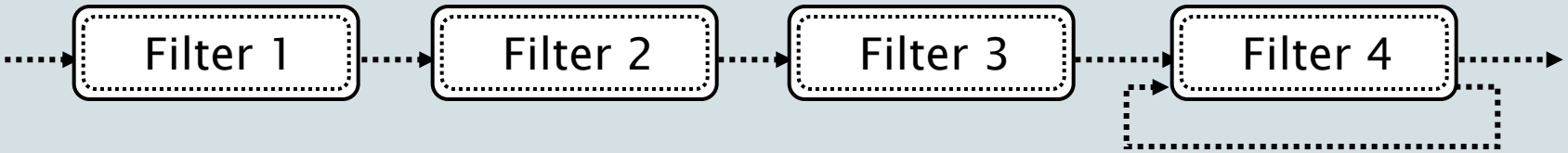
Special types of filters (?)

- **Pump (Producer/Source)**
  Produces data and puts it to an output port that is connected to the input end of a pipe.

- **Sink (Consumer)**
  Gets data from the input port that is connected to the output end of a pipe and consumes the data.



Pipe and Filter Style (2)



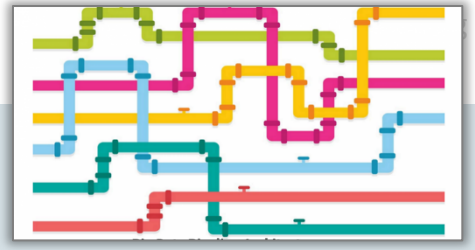
Constraints about the way filters and pipes can be joined:

- Unidirectional flow
- Control flow derived from data flow

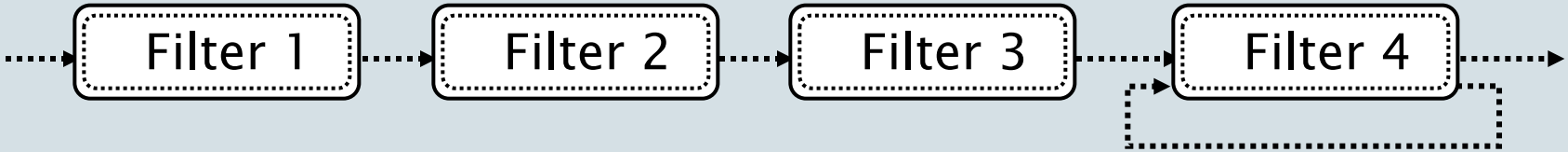
Behaviour Types:

a. **Batch sequential**
   Run to completion per transformation

b. **Continuous**
   Incremental transformation
   variants: push, pull, asynchronous



Pipe and Filter Style (3)



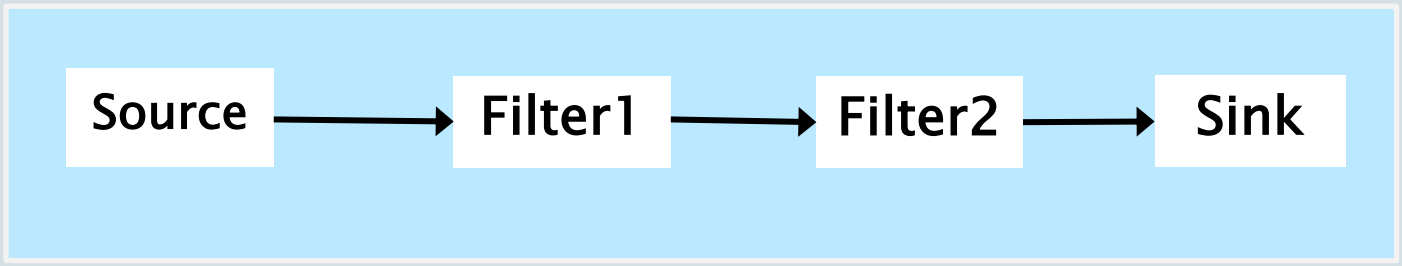
Semantic Constraints

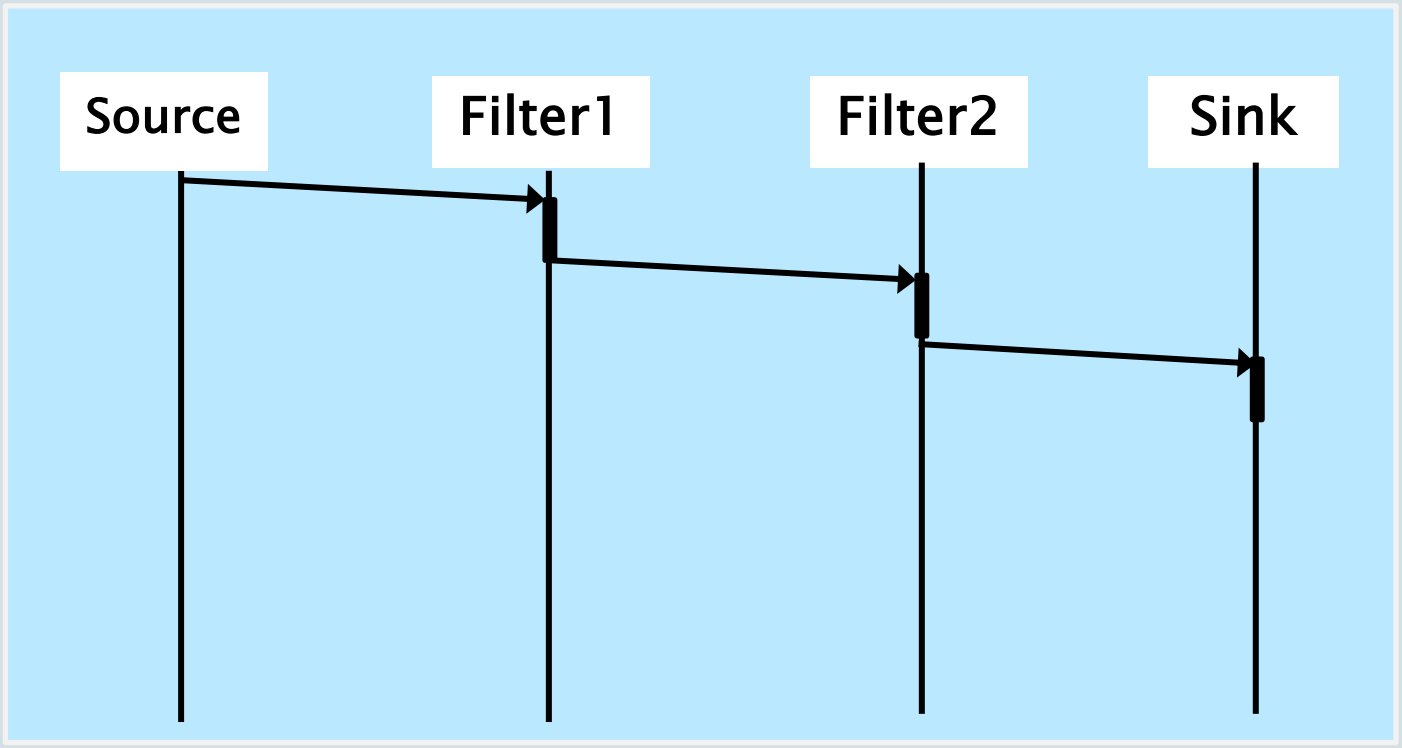
Filters are independent entities

- they do not share state
- they do not know their predecessor/successor

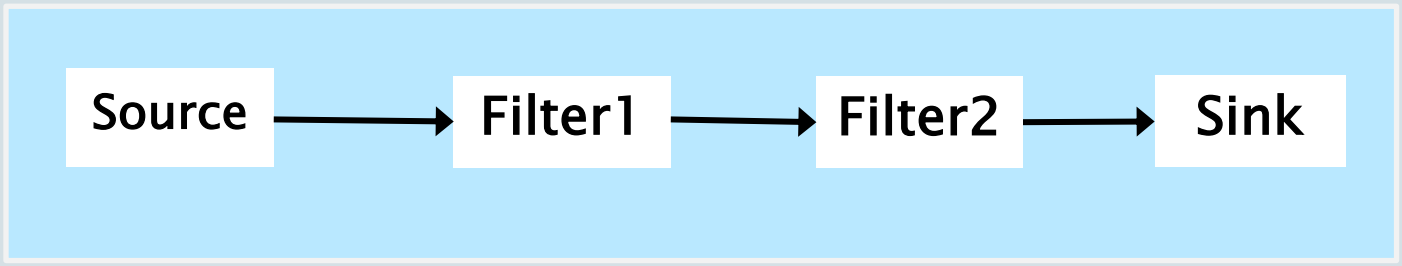
What are the dependencies between filters?
Compare this with Client Server?

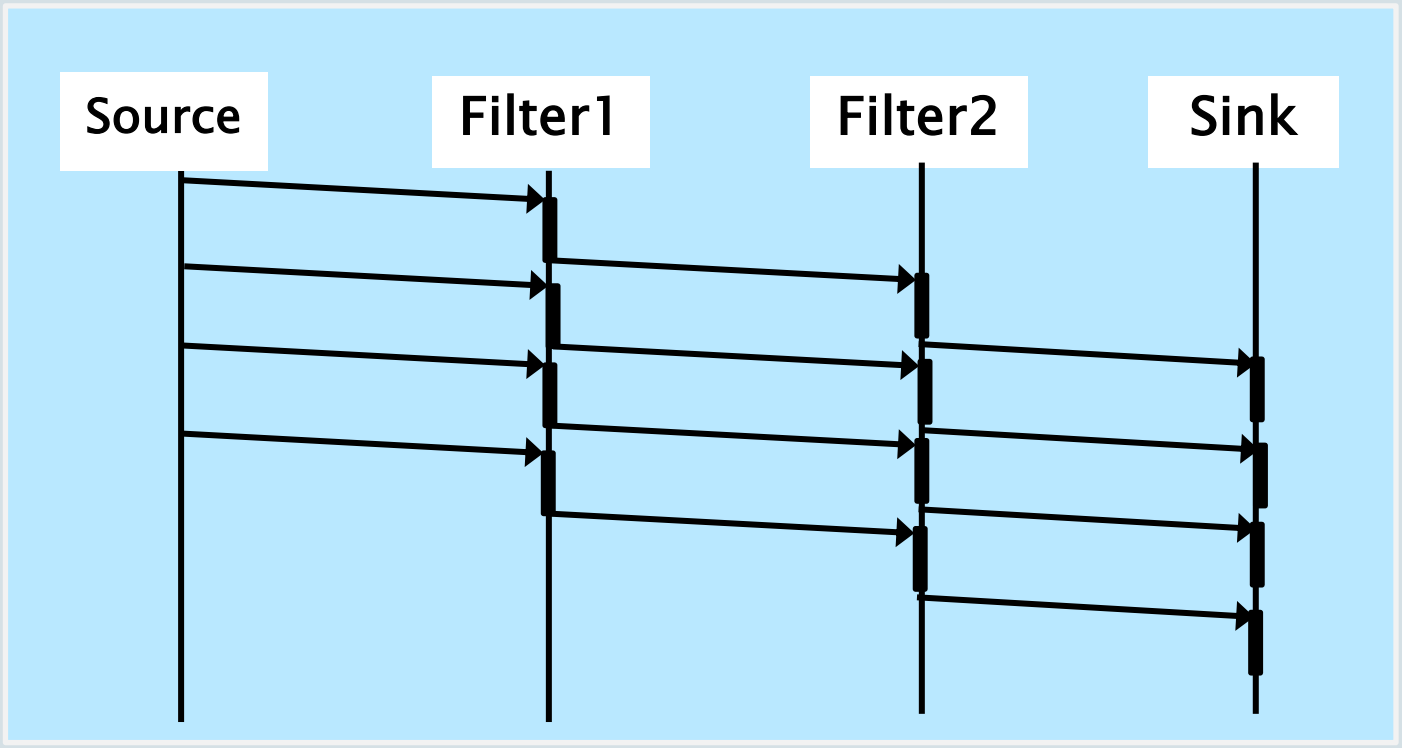
Pipe and Filter (Struct+Behaviour)



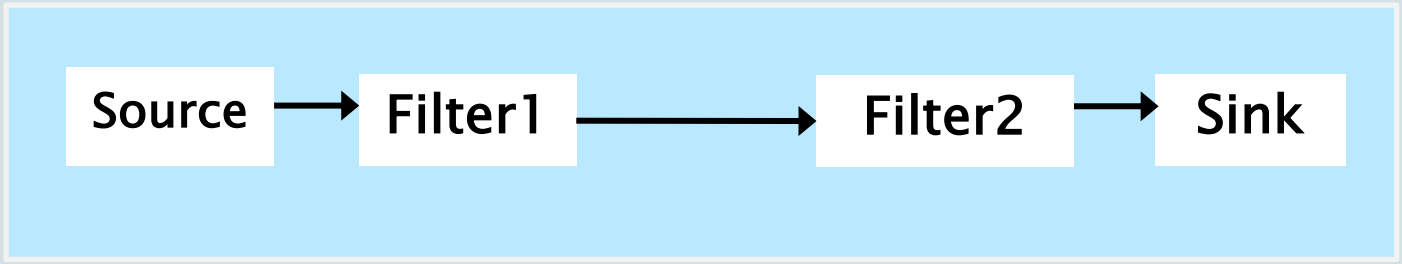


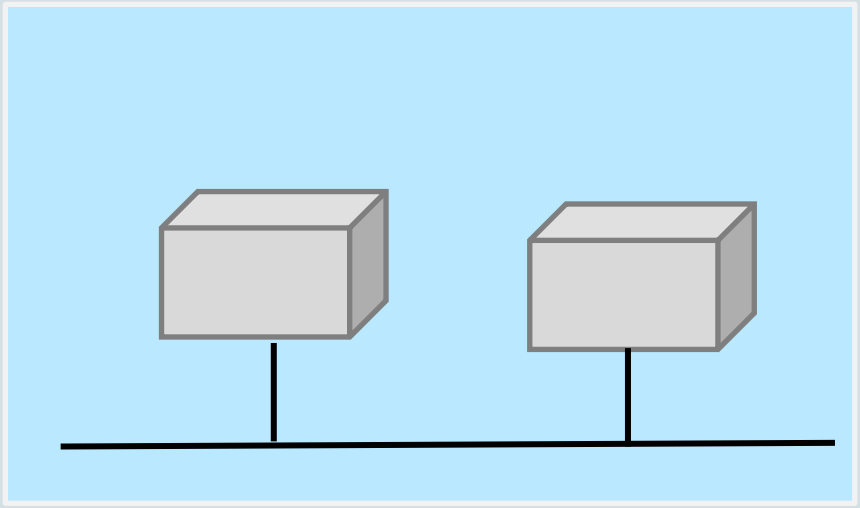
Pipe and Filter (Struct+Behaviour)





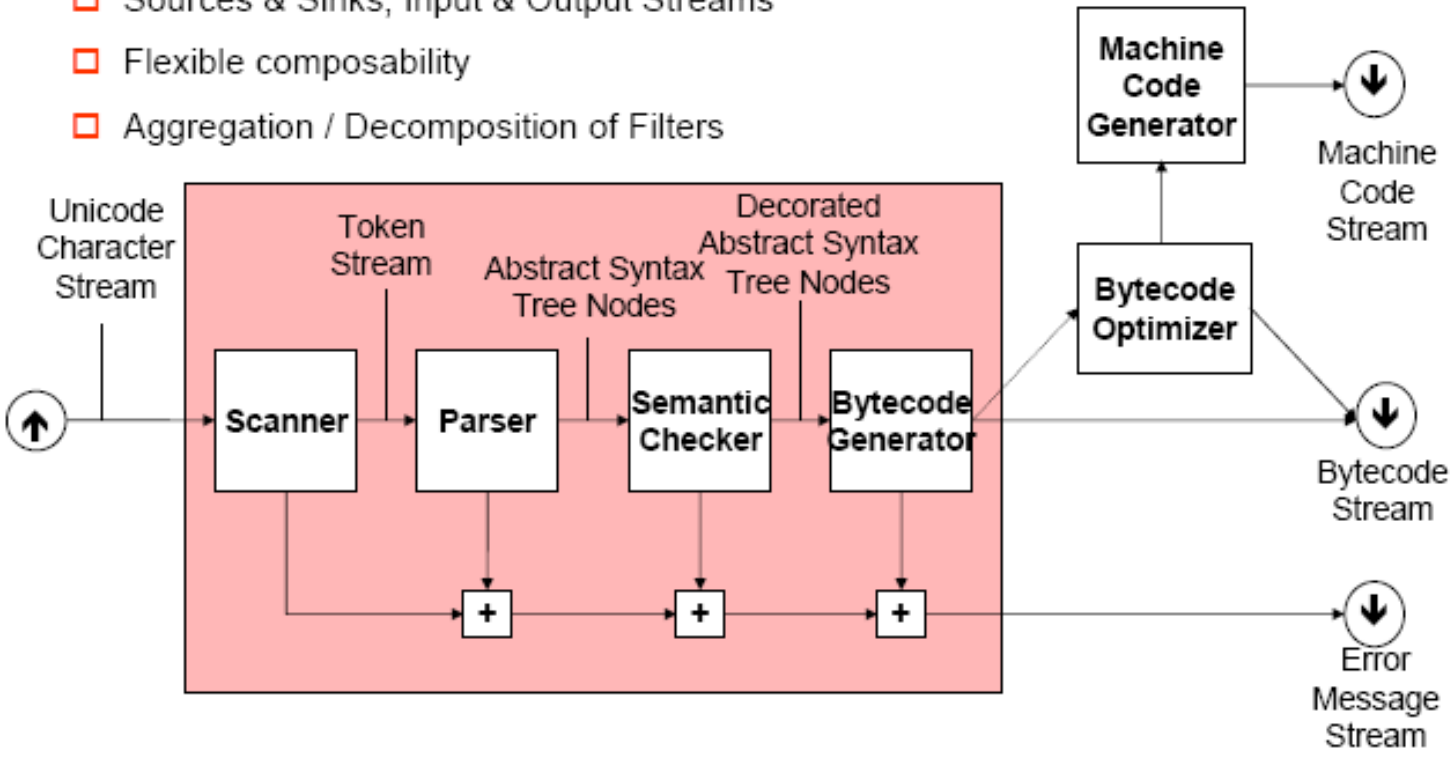
Pipe and Filter (Deployment)





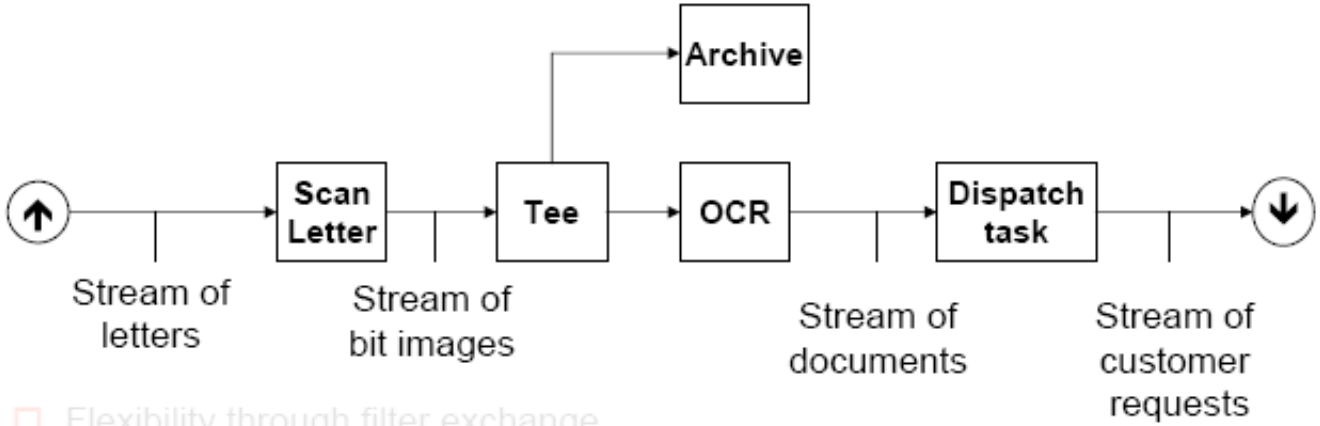
Example: P&F Compiler Architecture (1)

- Sources & Sinks, Input & Output Streams
- Flexible composability
- Aggregation / Decomposition of Filters



Example P&F Architecture

- No intermediate data structures necessary (but possible)
  (Pipeline processing subsumes batch processing)



- Flexibility through filter exchange
- Flexibility by recombination
- Reuse of filter components
- Rapid prototyping
- Parallel processing in a multiprocessor environment

Pipe and Filter Style (4a)

Advantages:

- Simplicity:
  - no complex component interactions
  - easy to analyze (deadlock, throughput, …)
- Easy to maintain and to reuse
- Filters are easy to compose (also hierarchically?)
- Can be easily made parallel or distributed

Pipe and Filter Style (4b)

Disadvantages:

- Interactive applications are difficult to create
- Filter ordering can be difficult
- Performance:
  - Enforcement of lowest common data representation, ASCII stream, may lead to (un)parse overhead
  - If output can only be produced after all input is received, an infinite input buffer is required (e.g. sort filter)
- If bounded buffers are used, deadlocks may occur

P&F Example: Linux commands

- ls | grep ‘architecture’ | sort
  - First ‘list files in directory’, then keep only files with ‘architecture’ in name, then sort this list

- ls | sort ls | grep ‘architecture’

This rearrangement works because components have the same input and output: the ‘lowest common denominator’ is a stream of lines of characters.

Pipe and Filter Style (5) Quality Factors

Extendibility: extends easily with new filters

Flexibility: – functionality of filters can be easily redefined,
– control can be re-routed
(both at design-time, run-time is difficult)

Robustness: 'weakest link' is limitation

Security: –

Performance: allows straightforward parallelisation

Pipe and Filter Style (6) Application Context

Rules of thumb for choosing pipe-and-filter (o.a. from Shaw/Buschman):

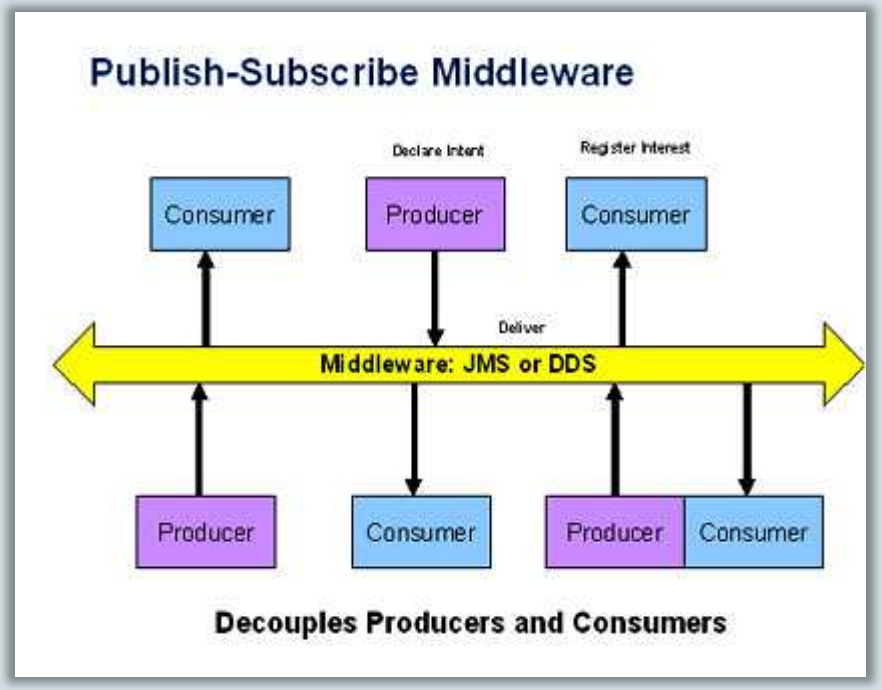
- if a system can be described by a **regular interaction pattern** of a collection of processing units at the same level of abstraction; e.g. a series of incremental stages (horizontal composition of functionality);
- if the computation involves the **transformation of streams of data** (processes with limited fan-in/fan-out)

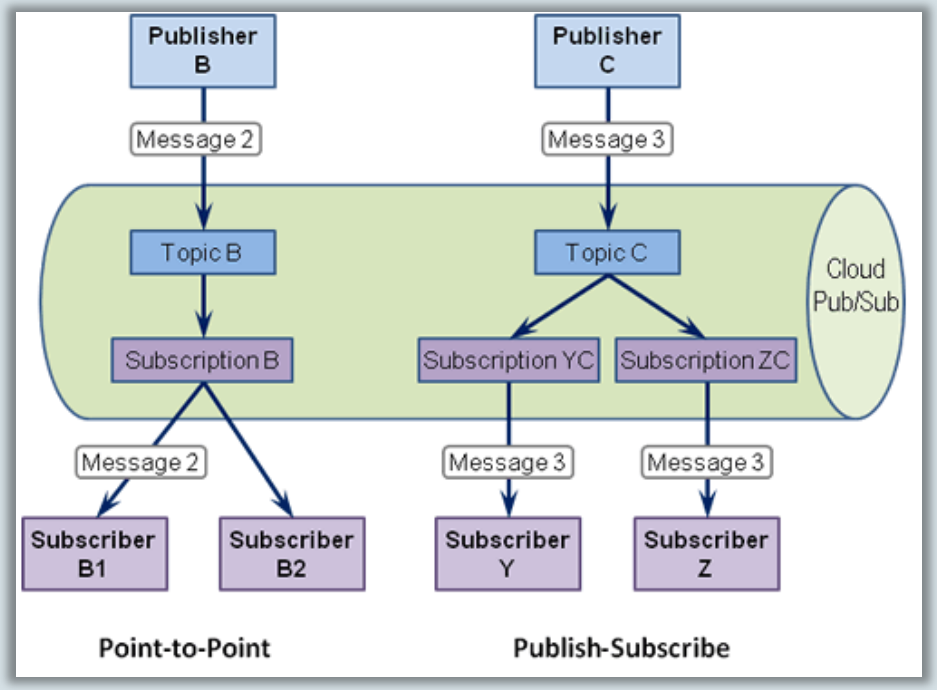
Hint: use a looped-pipe-and-filter if the system does continuous controlling of a physical system

Typical application domain: signal processing

CONTENTS

Publish-Subscribe

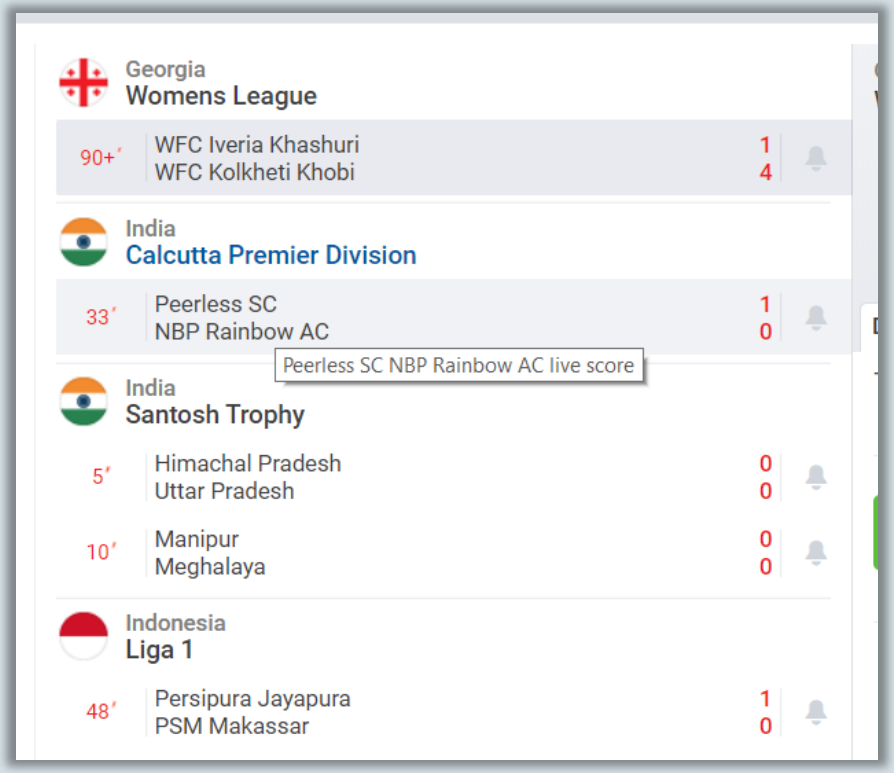




Publish-Subscribe

P/S is like: subscriptions that you know:

e.g. newspapers or live sports highlights:



Publish-Subscribe

- Components interact via announced messages, or events.
  - Components may subscribe to a set of events.
  - It is the job of the publish-subscribe runtime infrastructure to make sure that each published event is delivered to all subscribers of that event.

- **Advantages**: loose coupling, scalability, extendibility, improved security (messages sent to subscribers only)
- **Limitations**: need to guarantee delivery, performance problems when overloaded with messages

Publish-Subscribe Style
Case Study: SPLICE

Developed by Thales (formerly Hollandse Signaal App.)

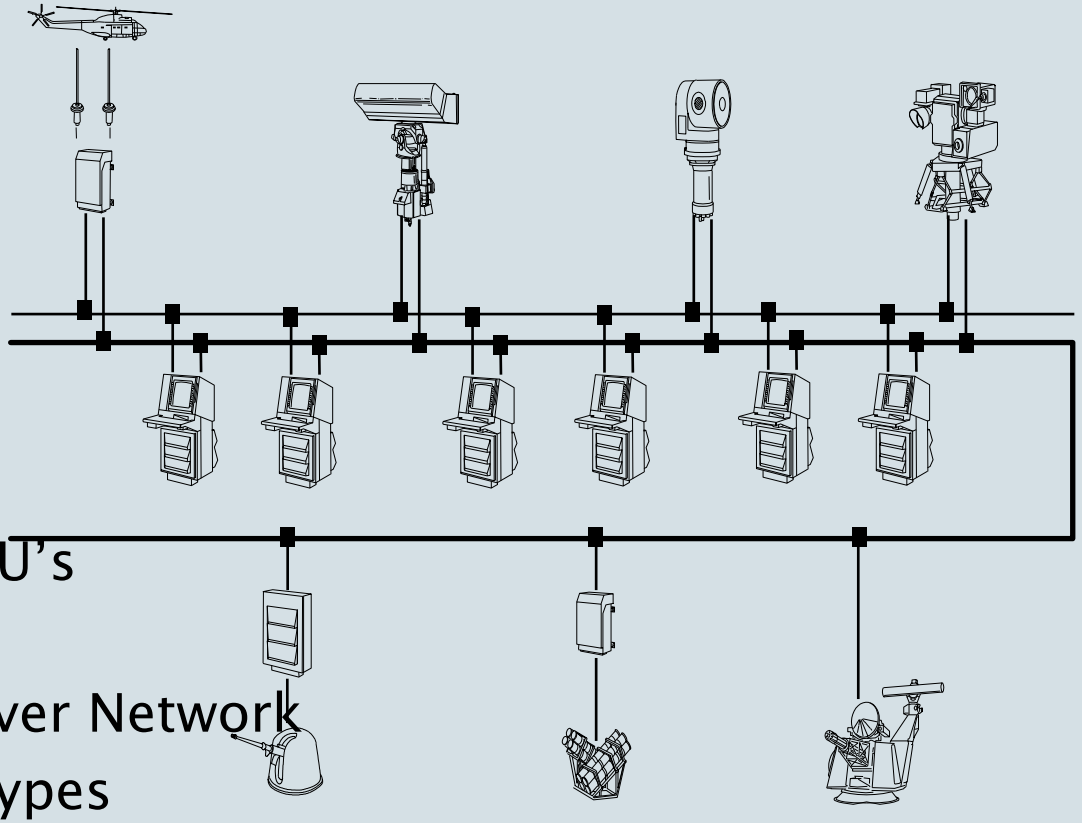
Oriented towards high quality control systems:

- Distributed
- Fault tolerant (support of degraded modes)
- (Soft) real-time
- Extensible

Architecture Requirements

The architecture is characterized as:

- real-time
- distributed
- data driven
- fault tolerant

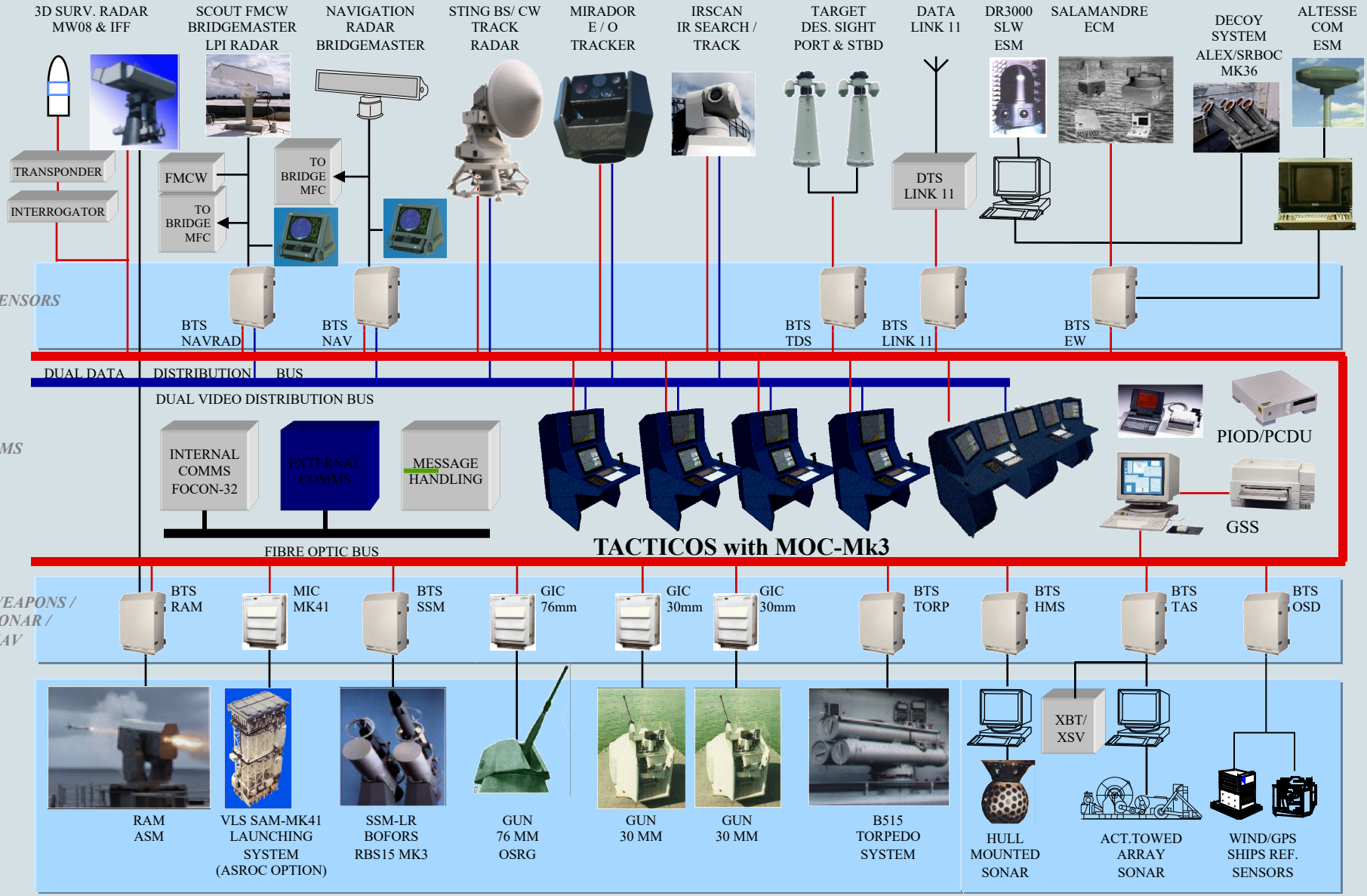


with some typical figures:

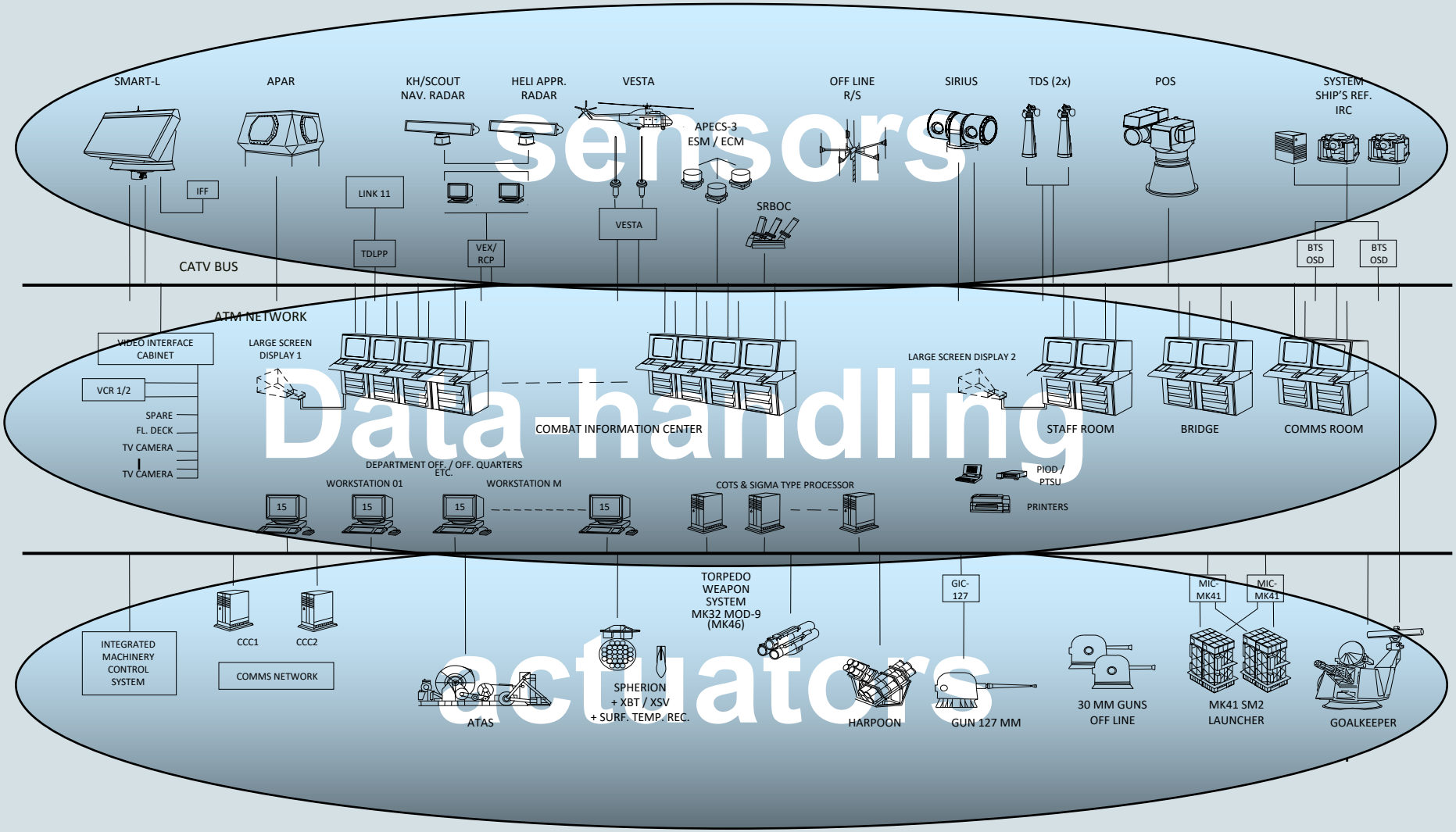
50 nodes containing 170 CPU's

- 2200 active executables
- 4000 Hz. data-updates over Network
- >2000 distributed data-types

Configuration example: frigate size system



Combat-Management-System Overview



SPLICE Application Domain

Used in command and control & traffic mgm. systems

Typical process:

1. **Acquire input**-signals through sensors
2. **Process input**-signals
3. **Interpret input** in terms of environment model
4. **Take action** through effectors
   or support operators in decision making

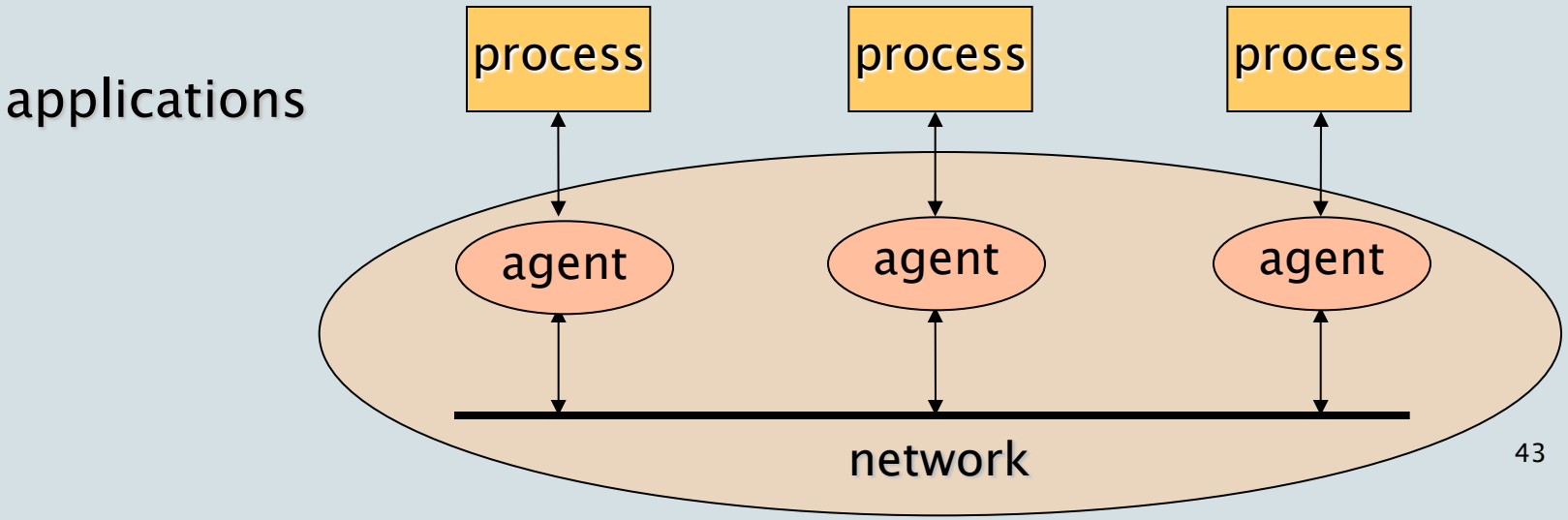
The interpretation of input may require the **sharing of data between many different applications** that **act in irregular patterns**

SPLICE Pub/Sub-Model

Applications are concurrently executing processes that implement part of the overall functionality

Processes register with network agents whether they are producers or consumers of a type of data.

The network agents manage distribution of data.



SPLICE Data Sorts

- Data elements are labeled records.
- Each record has a system-wide unique label, called the *data sort*
- A field of a sort may be declared ***key*** if it uniquely determines the values of the non-key fields

sort *flightplan*

key *flightnumber*	:	string
Departure	:	time
Arrival	:	time
Aircraft	:	string

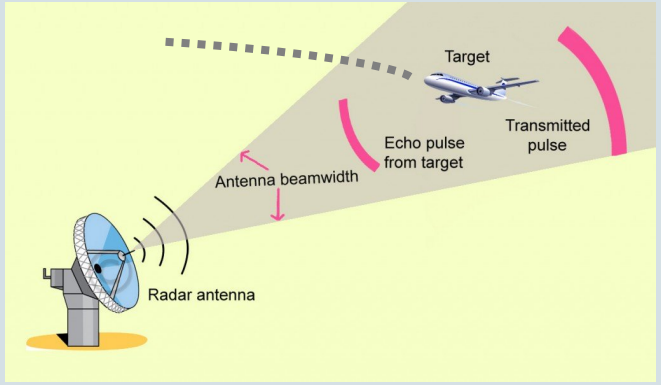
sort *track*

key *flightnumber*	:	string
key *index*	:	integer
State	:	string

SPLICE Example (1 /5)

Consider a system for tracking flying objects:

- Observations are made by a radar (and are called plots), i.e. the acquisition sensor
- Plots are correlated into tracks, that are interpreted in terms of a flight trajectory model
- Tracks are used to control the direction of the radar and for taking action through effectors)



SPLICE Example (2/5)

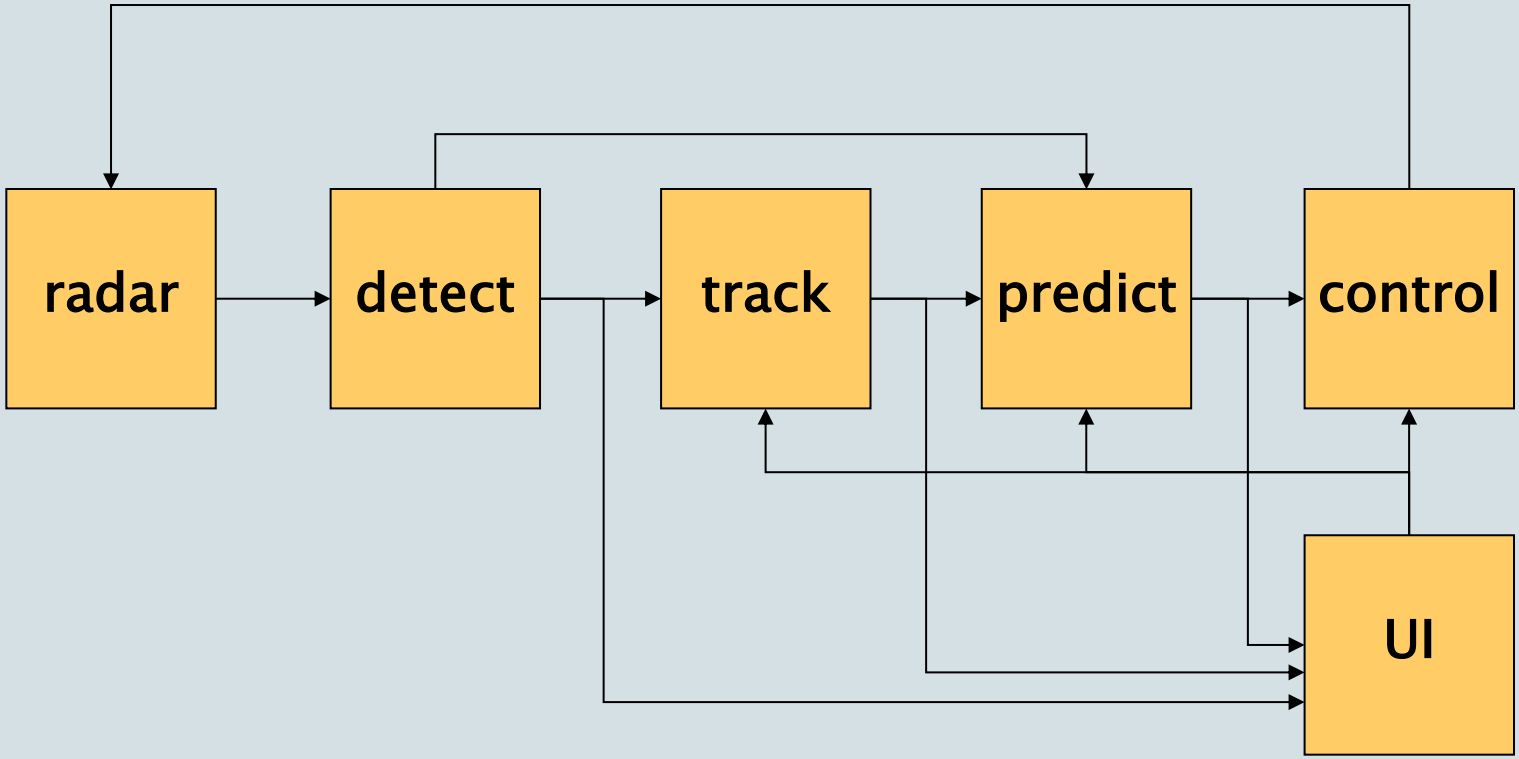
Processes

Radar:	generates signals
Detect:	processes radar signals into plots
Track:	correlates plots into tracks
Predict:	predicts next coordinates of the flying object
Control:	the radar to probe the next position of the object
UI:	user interface of the system

Sorts (=data types)

- D0: radar signals
- D1: control data to the radar
- D2: plots (coordinates) from the radar
- D3: sensor characteristics
- D4: speed-vector of the object
- D5: predicted object coordinates
- D6: user commands

Application model using one-to-one connections



SPLICE Example program (4/5)

```
Program Detect
   sort raw_data: radar_signals consumed
   sort obj_pos: coordinates produced
   signal: sensor_data

Forever do
    signal := get(raw_data);
    if valid_signal(signal)
        then obj_pos:= f(signal); put(obj_pos)
        else  { corrective action }
End program Detect
```

What happens when multiple copies of *Detect* are running concurrently?

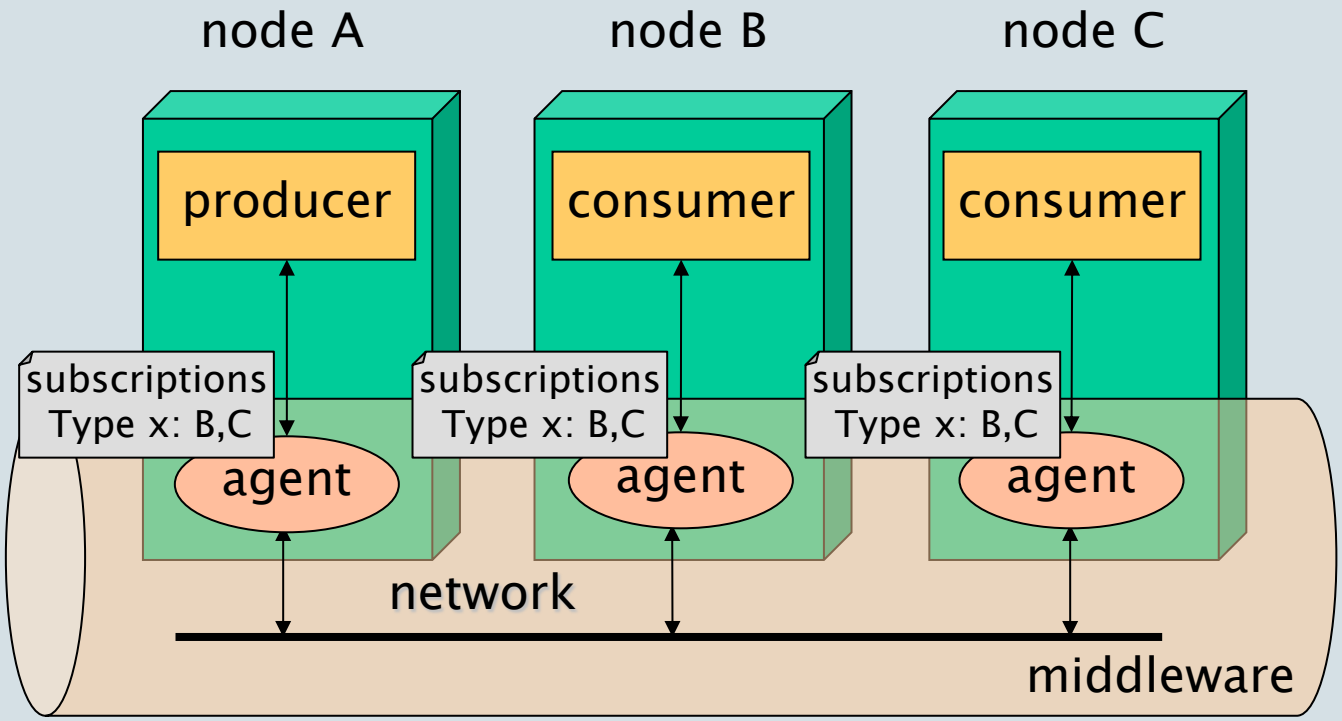
SPLICE Example (5/5)

```
Program Predict
  sort radar_attr: sensor_attr consumed
  sort track_data: track consumed
  sort pred_coord: coordinates produced
  sort user_cmnd: command consumed
  result: integer
  local_track: track

get(radar_attr);
Forever do
   result := get(track_data);
   if valid_track(result)
      then
         { local_track:=predict new coordinates};
         put(obj_pos)
      else
         if local_track.timestamp + radar_attr.cycle_time >
            time - comm_delay
         then { new data too late; corrective action };
   result := get(user_cmnd);
   if valid_cmnd(result)
       then  { deal with command }

End program Predict
```

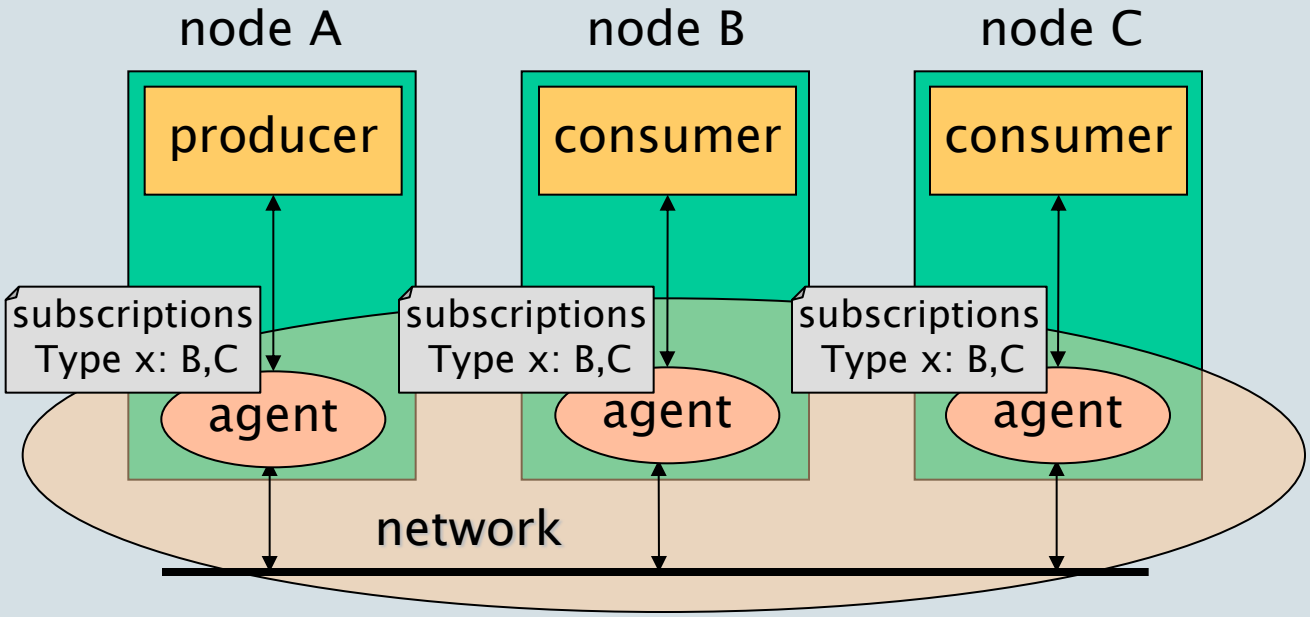
P/S Deployment



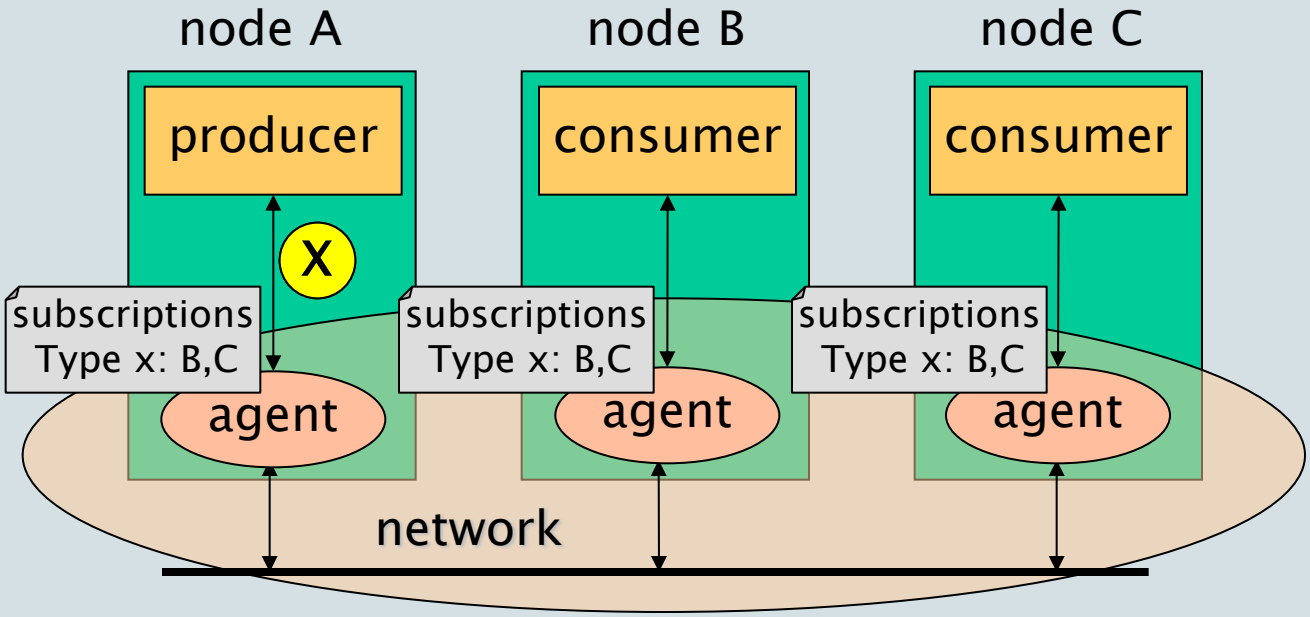
producer = application software component

agent = middleware software component

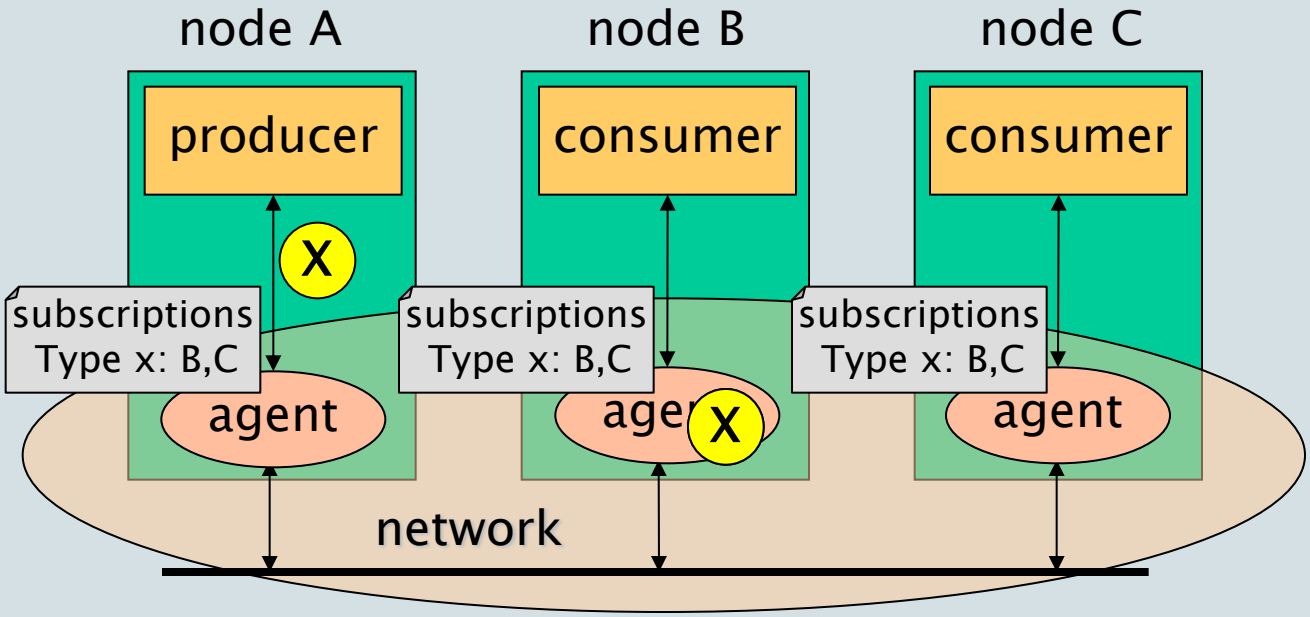
Registration Phase



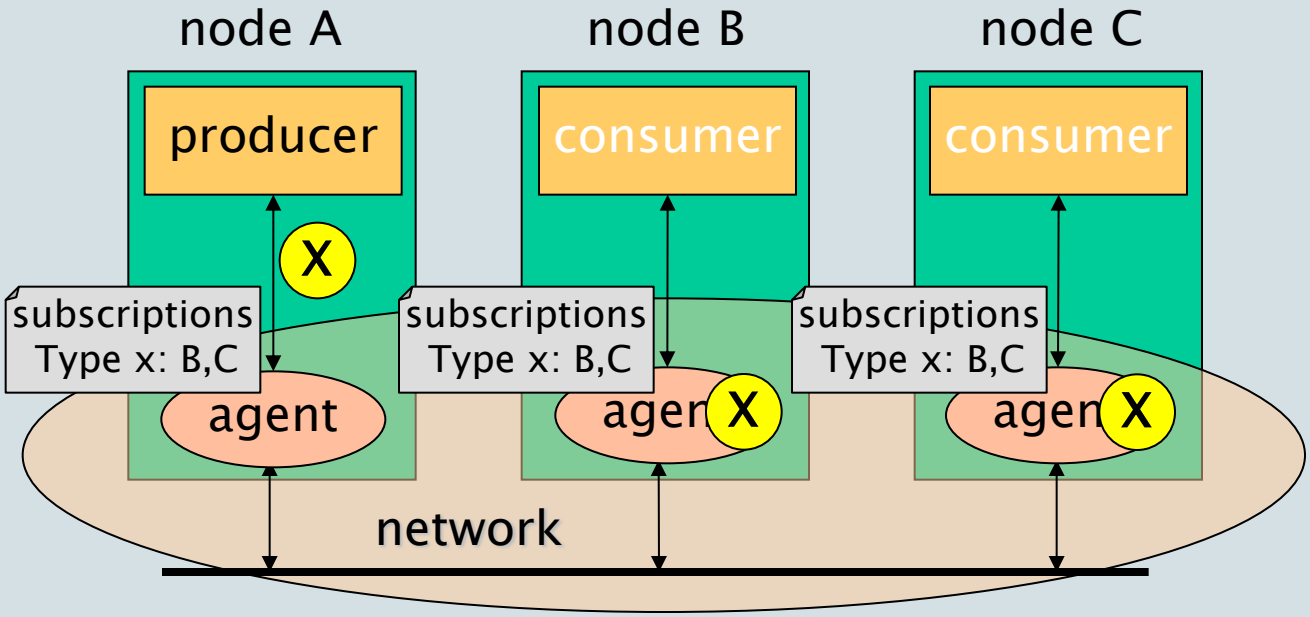
Distribution Phase



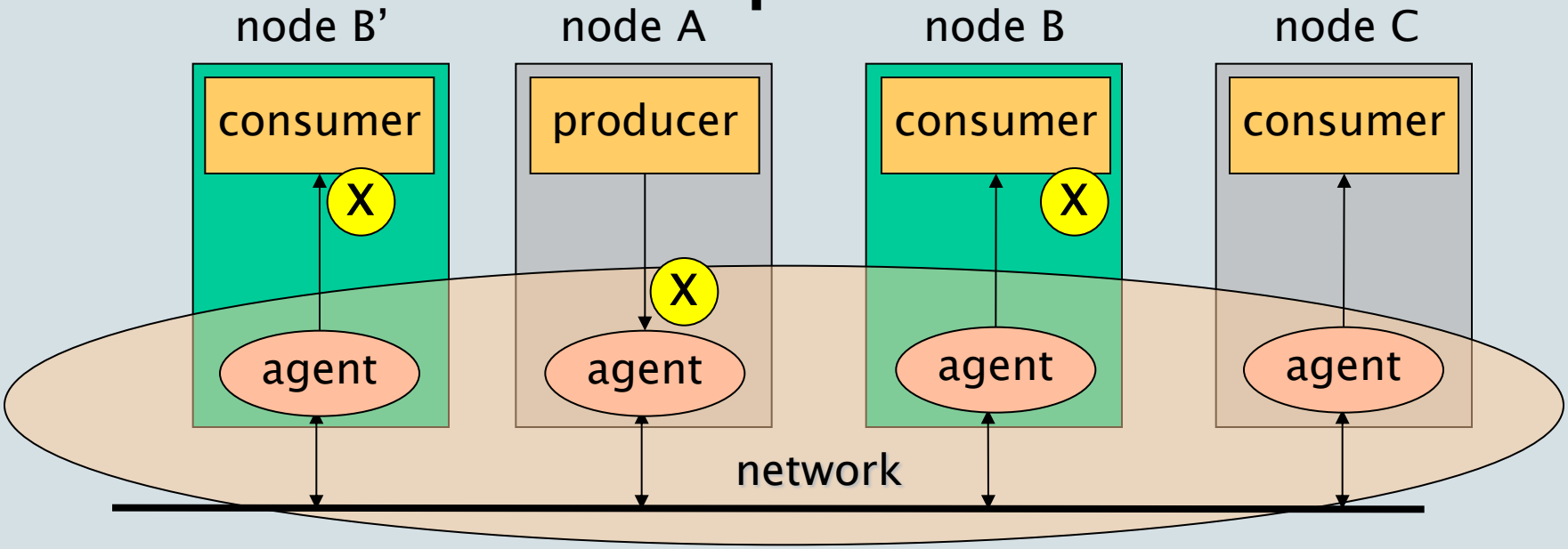
Distribution Phase



Distribution Phase



Insert New Consumer Component

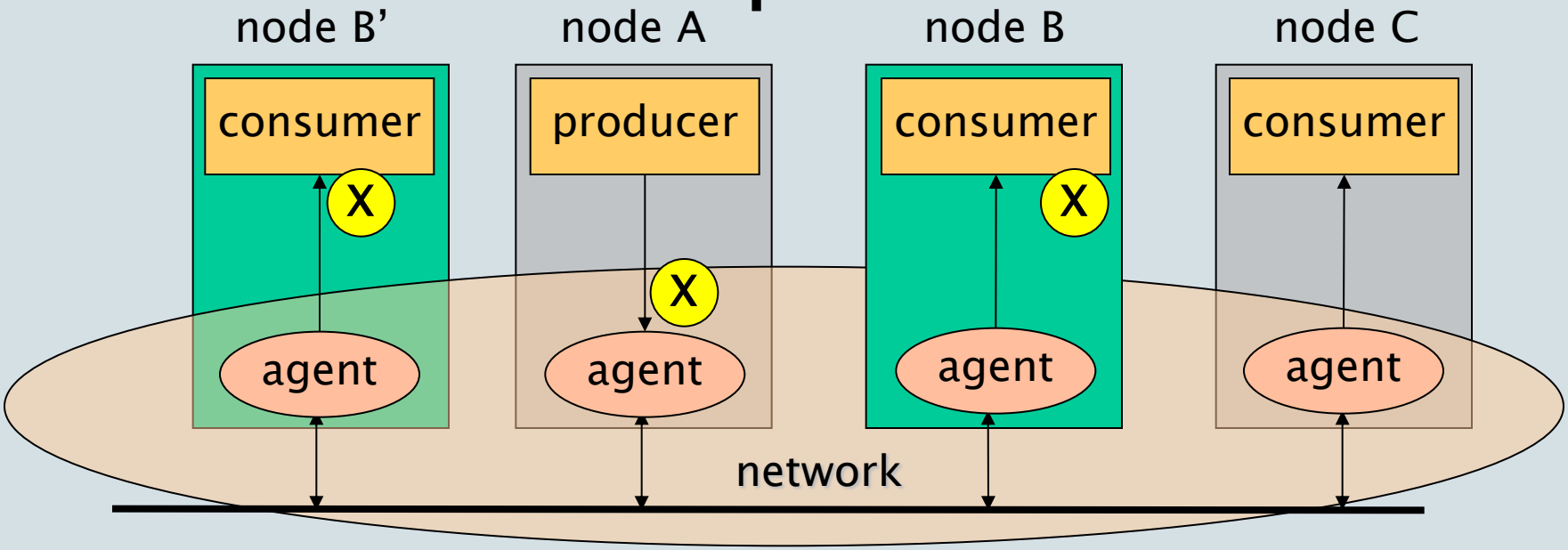


B' can build up state from inputs it receives.

If B and B' both consume and produce data, then duplicate data is generated.

B' can monitor output of B to check convergence

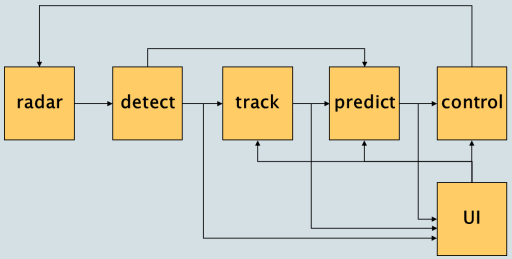
Phase out old Consumer Component

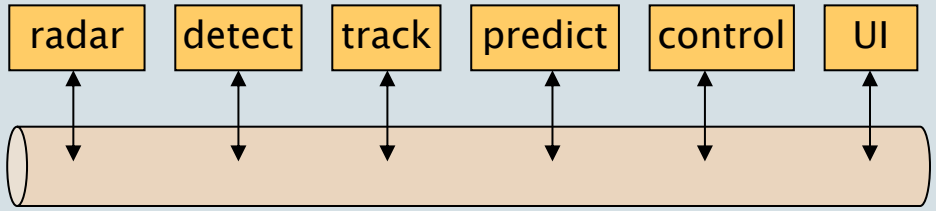


Once B' has converged with B, B is stopped

Reflection on Architectural Style of Pub/Sub

The architectural style strongly influences
- the complexity of the overall design, and
- the systems' quality attributes





When to use P/S

- Data is short-lived
- 'Frequent' production of data
- Consumers are interested in updates
- Multiple consumers
- Dynamically changing topology of producers and/or consumers

References

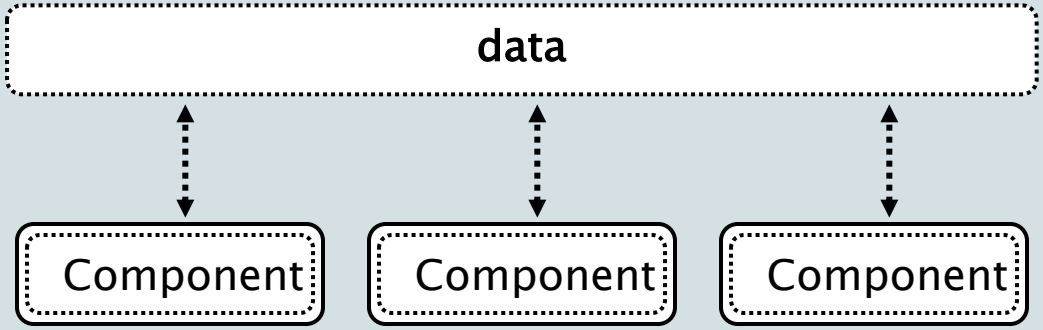
Control System Software, M. Boasson
IEEE Transactions on Automatic Control, Vo. 38, No. 7, July 1993

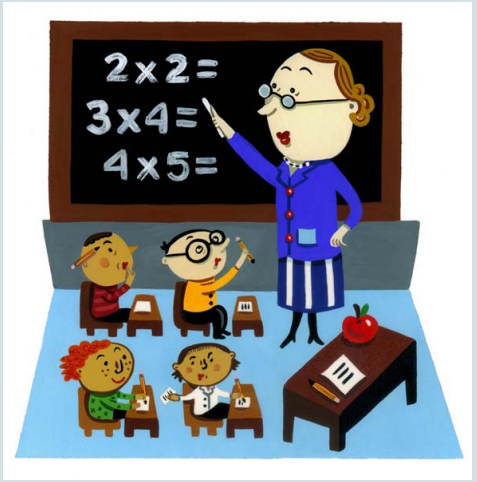
Software Architecture for Large Embedded Systems
M. Boasson and E. de Jong
http://www.cwi.nl/~marcello/SAPapers/BJ97.html

CONTENTS

Blackboard Style (1)

Concept: Concurrent transformations on shared data





Components: processing units (typically knowledge source)

Connectors: blackboard
interaction style: asynchronous

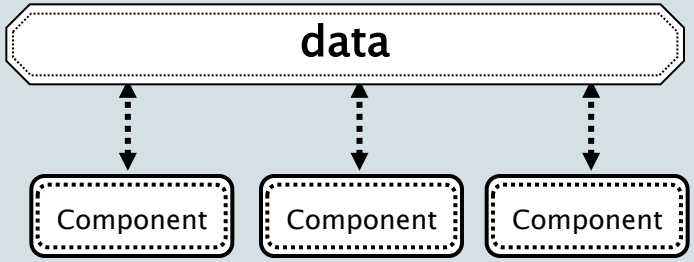
Topology: one or more transformation-components may be connected to a data-space,
there are typically no connections between processing units (bus-topology)

Blackboard Style (2)

Behaviour Types:

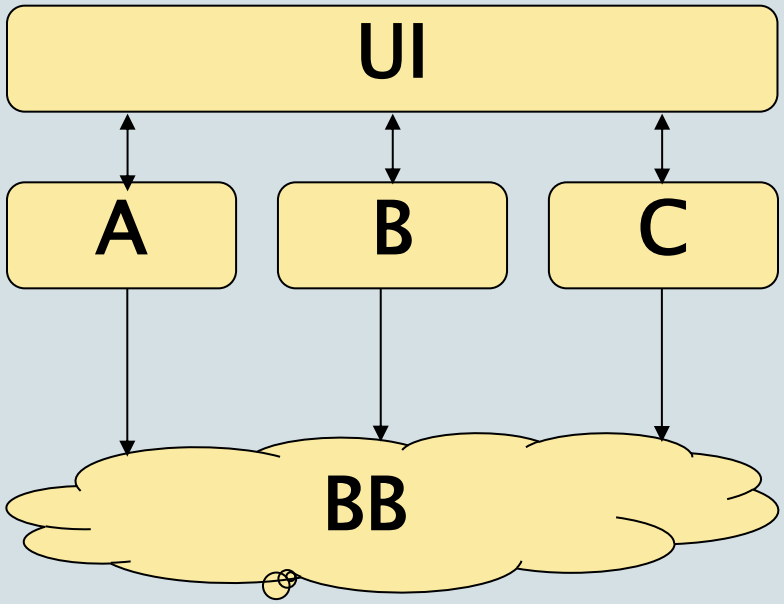
a. **Passive repository**
Accessed by a set of components; e.g. database or server

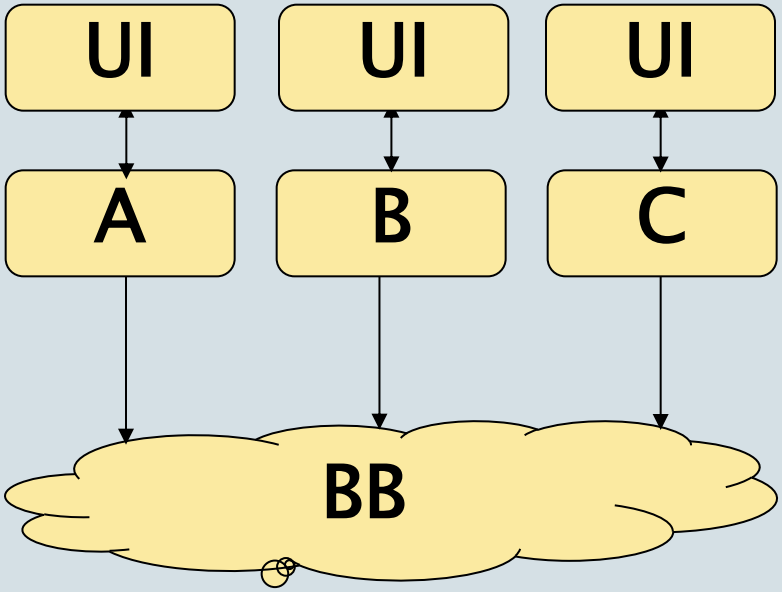
b. **Active repository**
Sends notification to components when data of interest changes; e.g. blackboard or active database



Constraints:
Consistency of repository: Various types of (transaction) consistency

Layering & Blackboard





Blackboard Style (3)

Advantages:

- Allows different control heuristics
- Reusable & heterogeneous knowledge sources
- Support for fault tolerance and robustness by adding redundant components

+/- Dataflow is not directly visible

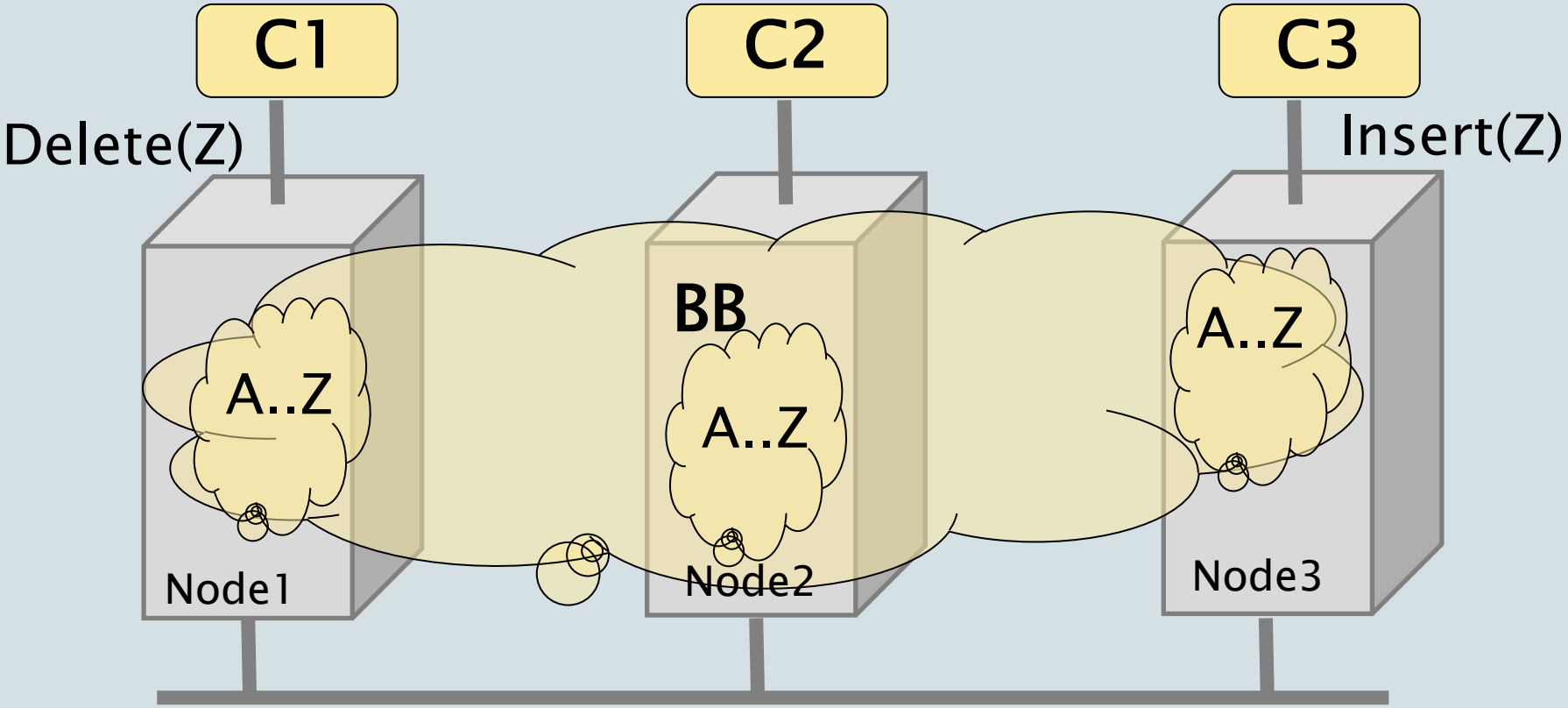
Disadvantages

- Distributed implementation is complex
  - distribution and consistency issues

Blackboard Characteristics

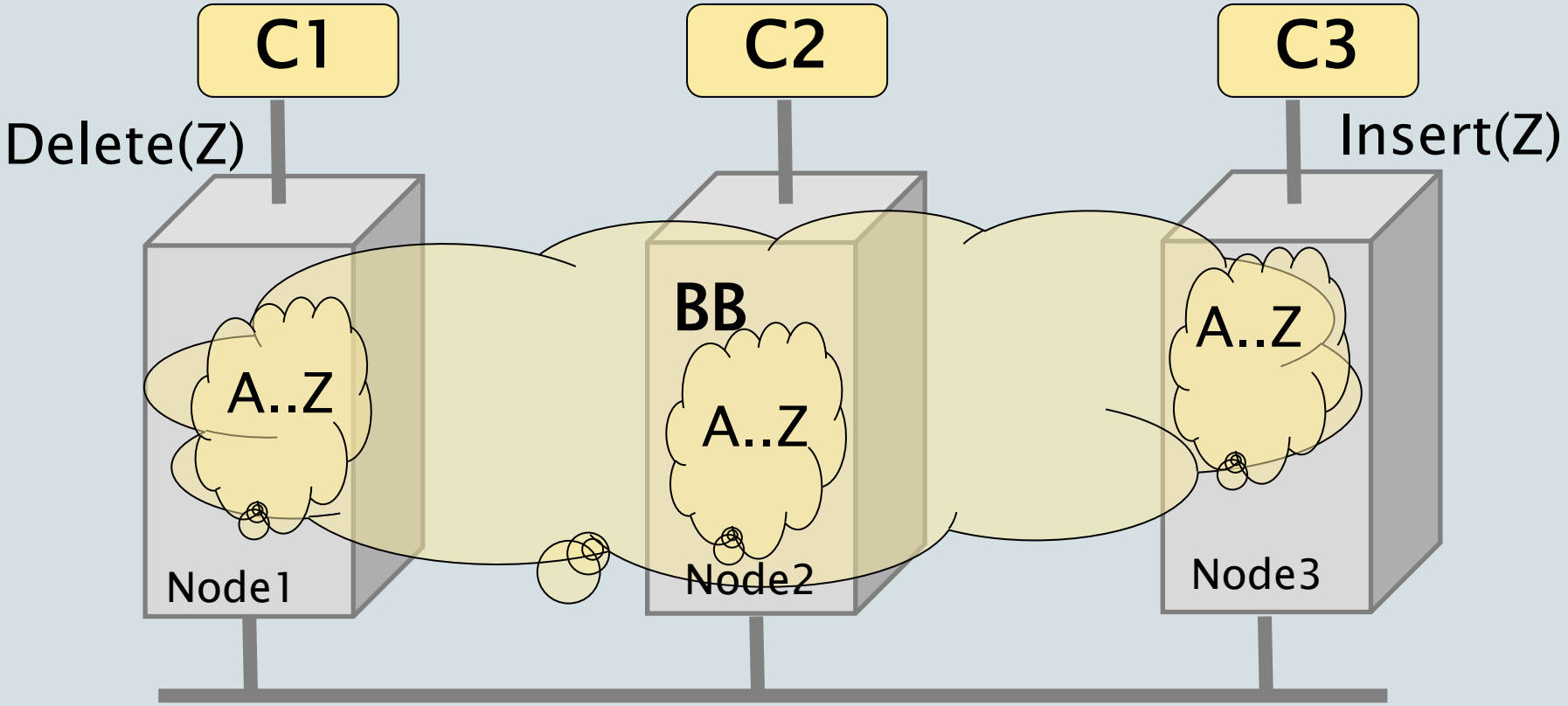
- Data may be structured (DB) or unstructured
- Data may be selected based on content
- Applications may insert/retrieve different data-type per access.
  This in contrast to pub-sub where data of the same type is retrieved repeatedly

Blackboard and consistency



Node 1, 2 and 3 are all storing a copy of the entire dataset (A-Z). This increases reliability & availability and improves response time *(. But ….

Blackboard and consistency



C1 and C3 may 'see' a different content on the blackboard depending on the order (and speed) of executing the delete and insert actions.

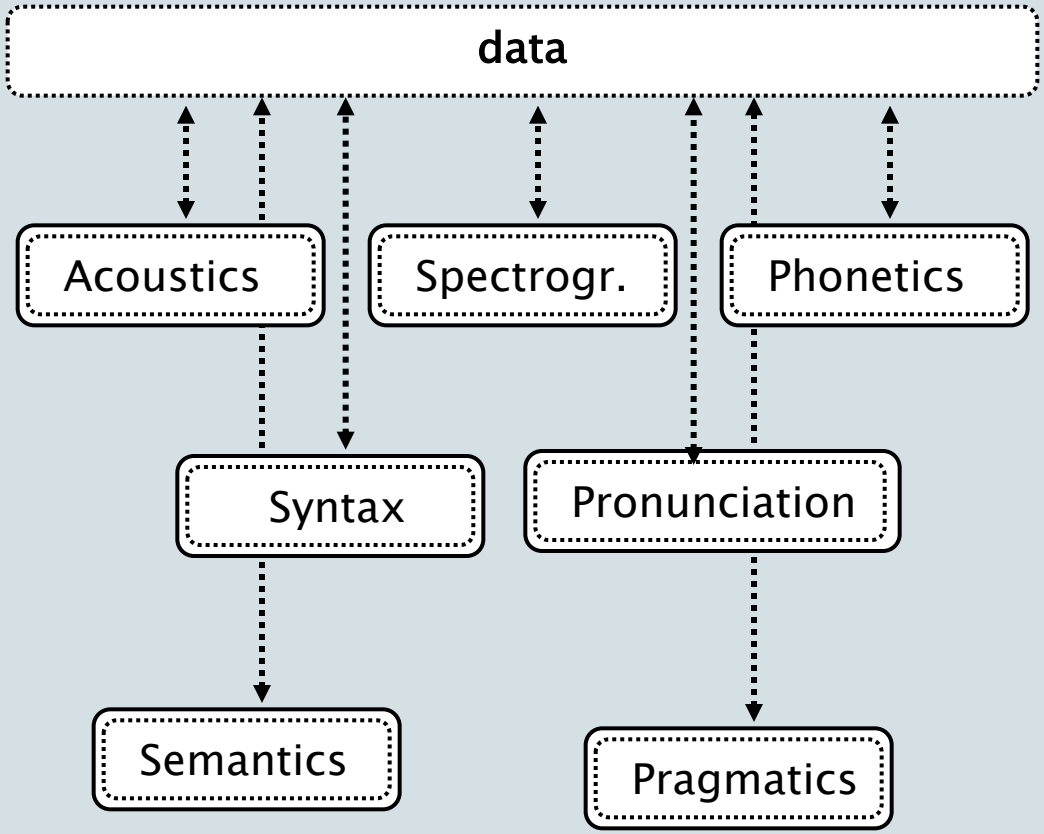
Example of Blackboard Architecture

- Hearsay, speech understanding
- Hearsay was developed in the 1970's by Raj Reddy et al. at Carnegie Mellon University.
- Randy Davis, *Speech Understanding Using Hearsay,* MIT videotape, 1984.

Slides adapted from Terry Bahill, Univ. Arizona, 2007

Hearsay: knowledge sources

- Acoustics
- Spectrographs
- Phonetics
- Pronunciation
- Coarticulation
- Syntax
- Semantics
- Pragmatics



Hearsay: levels of abstraction*

Sentences

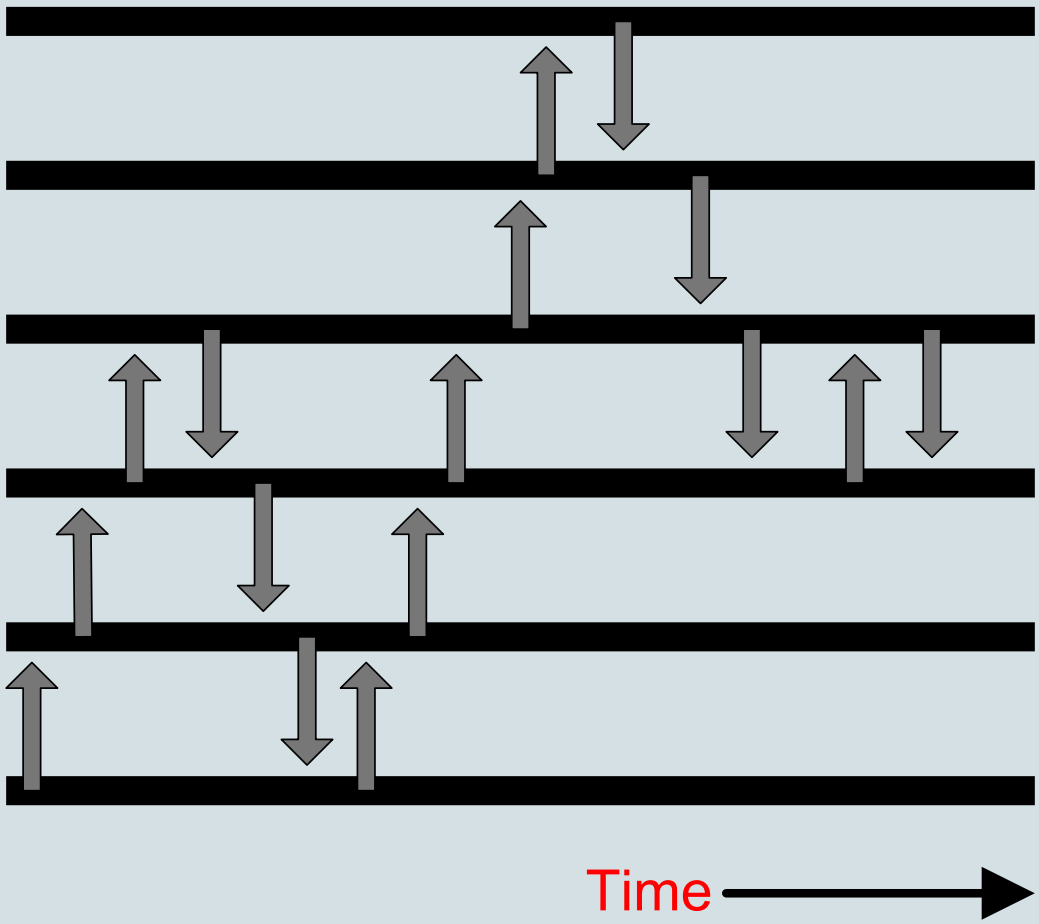
Phrases

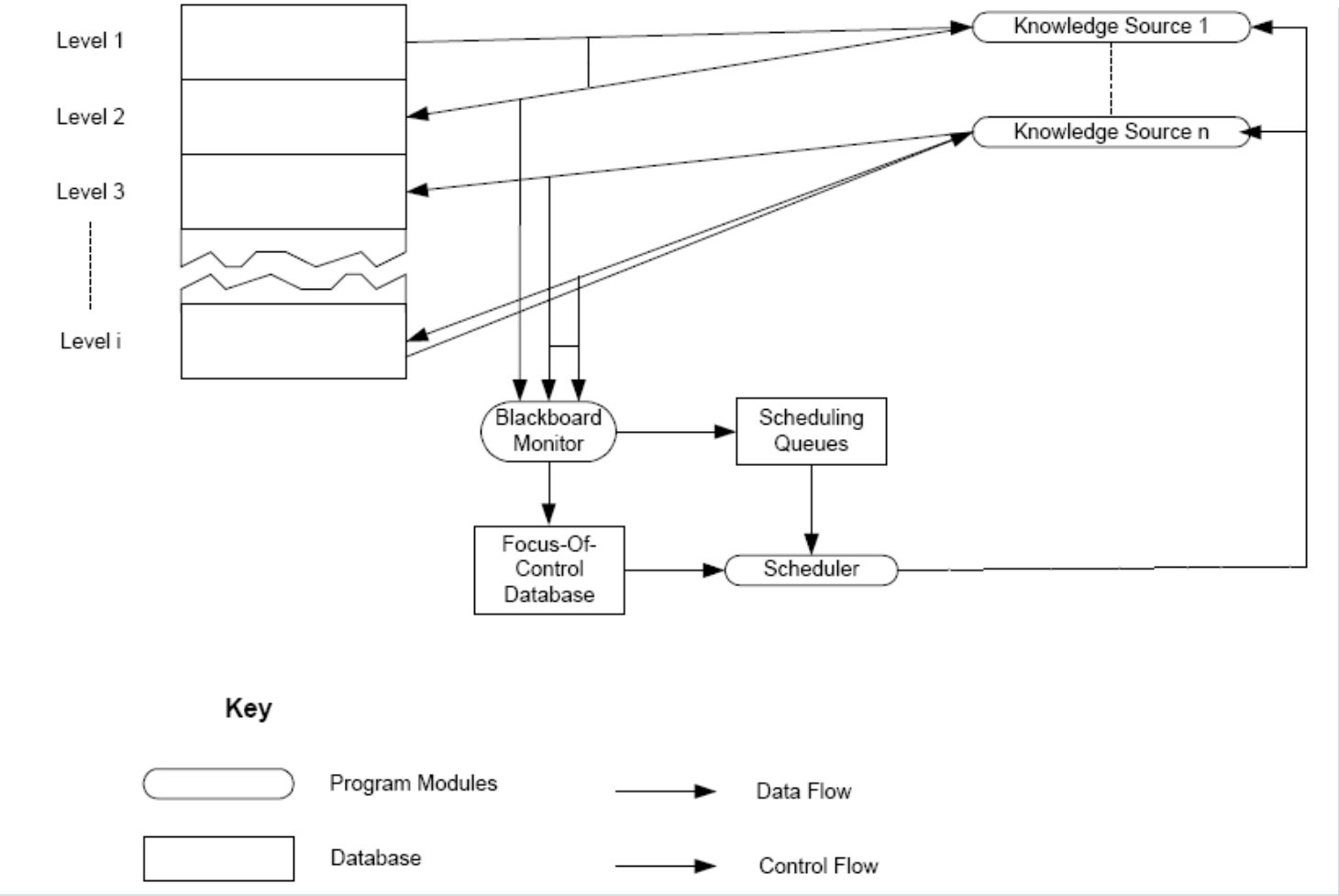
Words

Syllables

Phonemes

Acoustic waveform





- L.D. Erman, F. Hayes-Roth, V.R. Lesser and D. R. Reddy, "The Hearsay-II speech understanding system: integrating knowledge to resolve uncertainty", ACM Computing Surveys 12(2), pp213-253, 1980.
- L.D. Erman, P.E. London and S. F. Fickas, "The Design and an Example Use of Hearsay-II", Proc. IJCAI-81, pp 409-415, 1981.

Hearsay: control

- Data driven
- Asynchronous
- Opportunistic
- Islands of reliability
- Combined top-down and bottom-up

Blackboard Style (4) Quality Factors

Extensibility: components can be easily added

Flexibility: functionality of components can be easily changed

Robustness: + components can be replicated,
- blackboard is single point of failure

Security: - all process share the same data
+ security measures can be centralized around blackboard

Performance: easy to execute in parallel fashion
consistency may incur synchroniz.-penalty

Blackboard Style (5) Application Context

Rules of thumb for choosing blackboard (o.a. from Shaw):

- if representation & management of data is a central issue
- if data is long-lived
- if order of computation
  - can not be determined a-priori
  - is highly irregular
  - changes dynamically
- if units of different functionality (typically containing highly specialized knowledge) concurrently act on shared data (horizontal composition of functionality)

Example application domain: expert systems

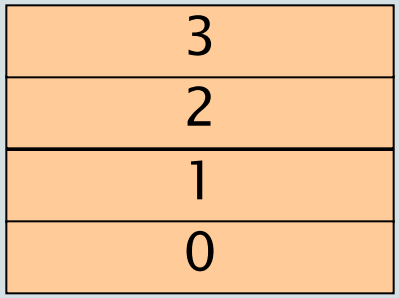
CONTENTS

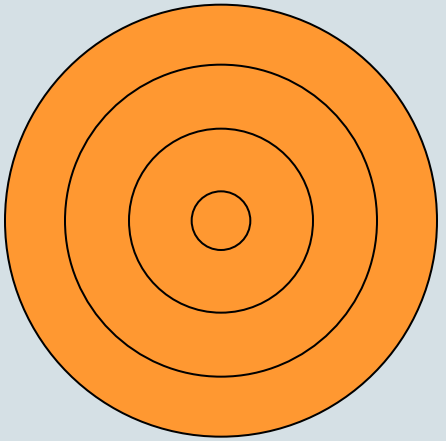
Layering (1)

Goals: Separation of Concerns, Abstraction, Modularity, Portability

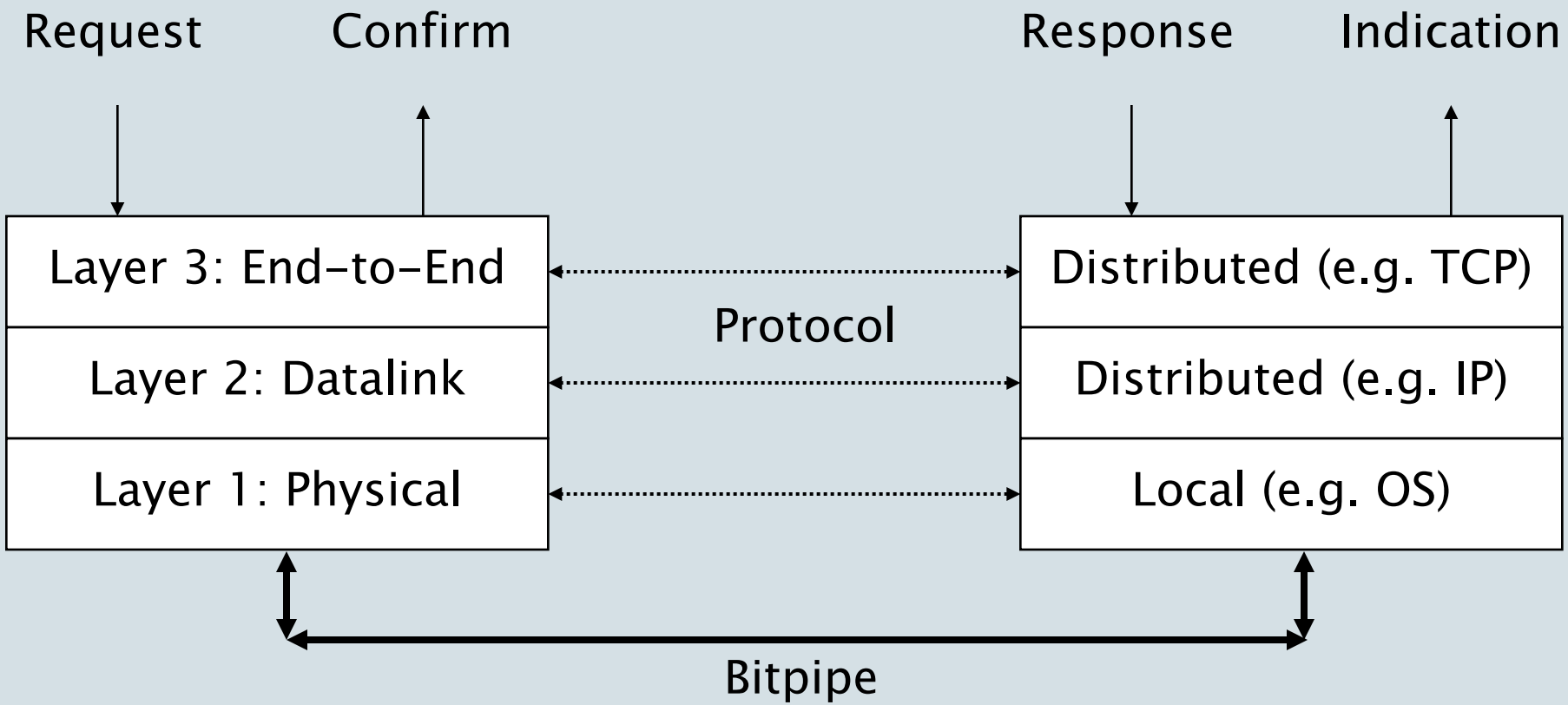
Partitioning in non-overlapping units that

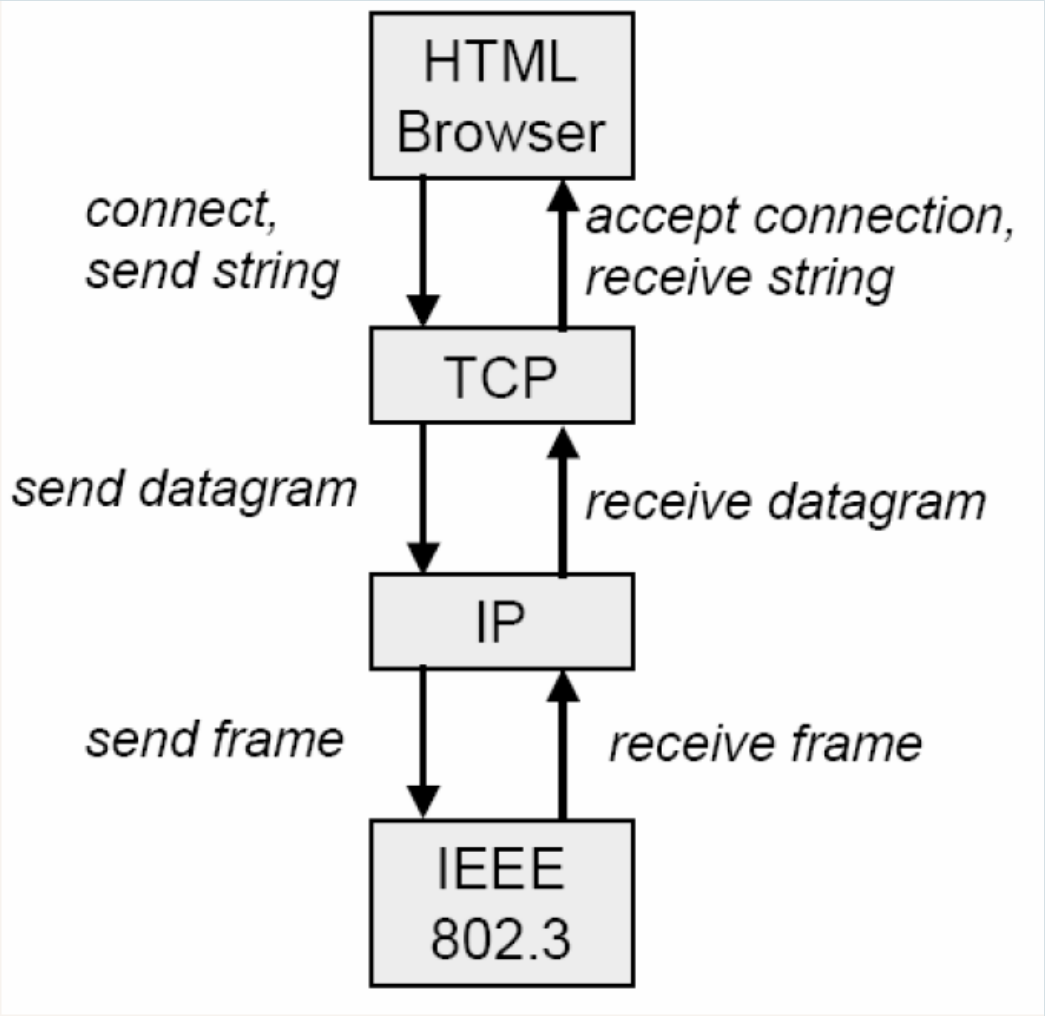
- provide a cohesive set of services at an abstraction level
  (while abstracting from their implementation)
- layer *n* is allowed to use services of layer *n-1*
  (and not vice versa)
  alternative:
  bridging layers: layer *n* may use layers $< n$
  enhances efficiency but hampers portability





Layering (2)
Example: Communication Stack





Picture from Jeremy Bradbury, Queens Univ. Canada

Layering (3)
Example: Virtual Machine

Concept: Separation of application lang. from execution platform

Emulation of a higher-level language

Types:

a. **Language interpreters**
   E.g. Java virtual machine, Unix command shell

b. **Rule-based systems**
   E.g. inference machine, expert system

Constraints: Defined by language

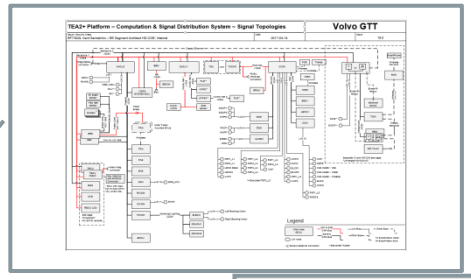
Advantages:

- Easy implementation
- Parts can be changed during execution

Disadvantages:

- Performance overhead

An example in Automotive Domain: Vehicle Monitoring & Control System



The main reasons:
1. Different nature and concerns
2. Differnt life cycles
3. Differnt kinds of complexities

Computation & Signal Distribution System

Basic Support Services/Utilities

Vehicle Application System

Device Monitoring & Control

...

Physical Perimeter System

Energy Supply & Distribution System

Torque Supply & Distribution System

Electricity Supply, Storage & Distr. System

Pneumatic Air Supply, Storage & Distr. Sys.

...

Door

Air Tank

Battery

Wheel

Differntial

Friction Brake

Ethernet Transceiver

CAN Transceiver

A/D Converter

Digital I/O Driver

Temperature Sensor

Function: Monitoring Air Inlet Pressure

Converts look and feel into pixels

Deals with the look and feel of the user interface instrument

Deals with WHAT shall be shown to a driver

A representation of Inlet Air and its charactersitics, i.e. convert measurement to a pressure unit like kPa.

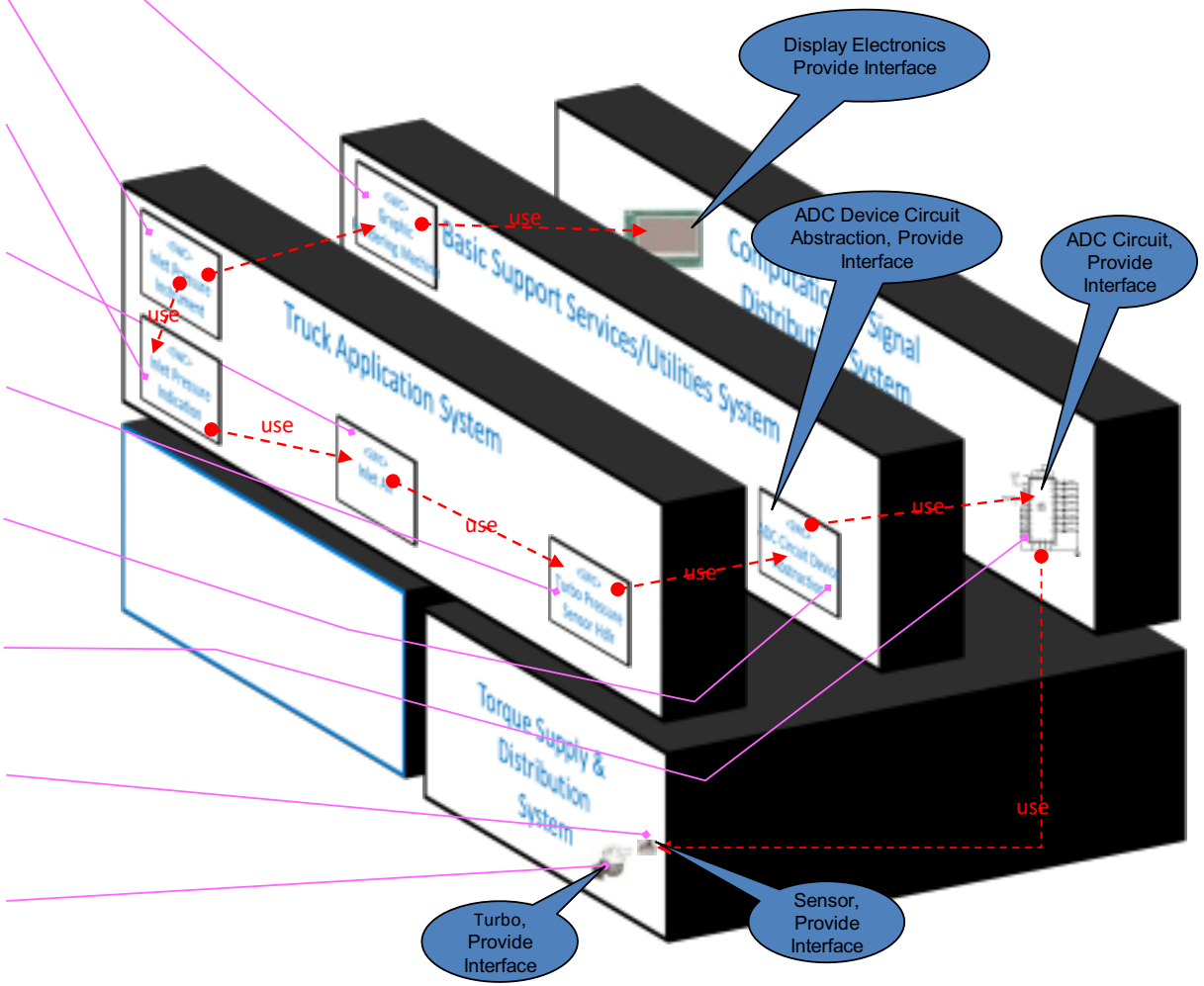
Deals with sensor characteristics, e.g. hysteresis, linerarity, ...

Access the A/D circuit, i.e. deals with circuit characteristics

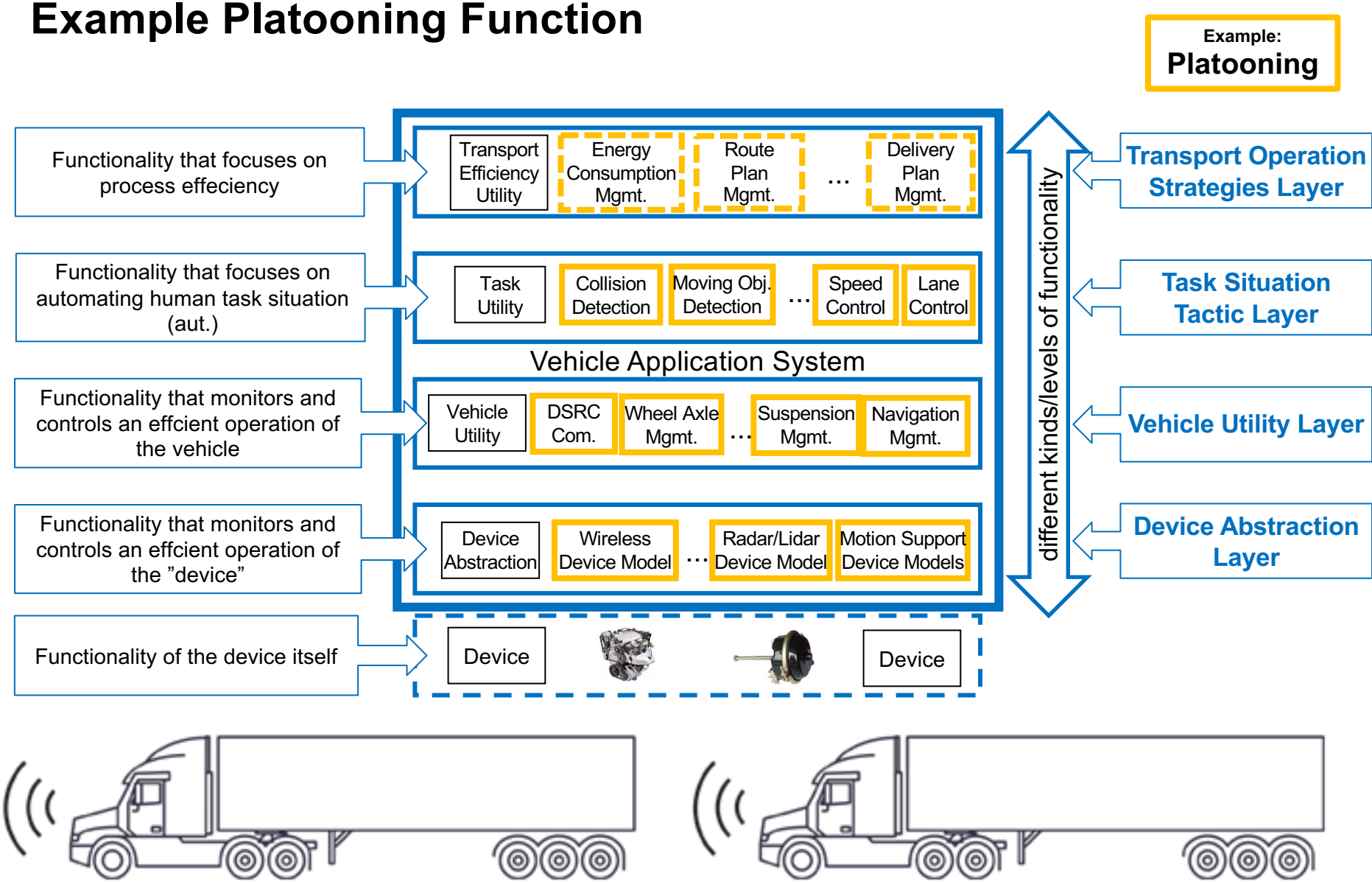
Convert an analogue value to a digital, change units

Convert a strain to a voltage level, change units

Increase pressure = change characteristics of the same unit



Layers in the Vehicle (Truck) Application Systems
Example Platooning Function



Layering (4) Quality Factors

Scalability:	n.a.*
Flexibility:	layers can be redefined
Robustness:	'weakest layer' is limitation
Security:	security measures should be taken at every layers' interface

To understand a system as a whole, the number of layers
Should be limited to an intellectually manageable number: ±7

Layering (5) Application Context

Rules of thumb for using layering:

- if data processing progresses through successive levels of abstraction
  (vertical composition of functionality)

Layering is a technique that helps in structuring systems

Typical examples: OS, device drivers, virtual machine (JVM), ISO, Client/Server

Division of Functionality

Pipeline:
- Multiple functional units operating in sequence (units chosen as steps in process)
- Regular pattern of computation for the class of inputs
- Functional units at same level of abstraction

Blackboard:
- Multiple functional units where order of operation is irregular or not know a-priori
- Allows concurrent operation of functional units
- Functional units at same level of abstraction (typically highly specialized processing)

Layered:
- Functionality (services) which are concerned with same level of abstraction are grouped

Summary Architectural Styles

Every Architect should have a standard set of architectural styles in his/her repertoire

- it is important to understand the essential properties of each style: when to (not) use them
- examples:
  - C/S, pipe and filters, blackboard, pub/sub, P2P

The choice for a style can make a big difference in the quality properties of a system

- analysis of the differences can provide rational for choosing a style

Questions ?