Software architecture

Service-Oriented Architecture

Jeremy Gibbons

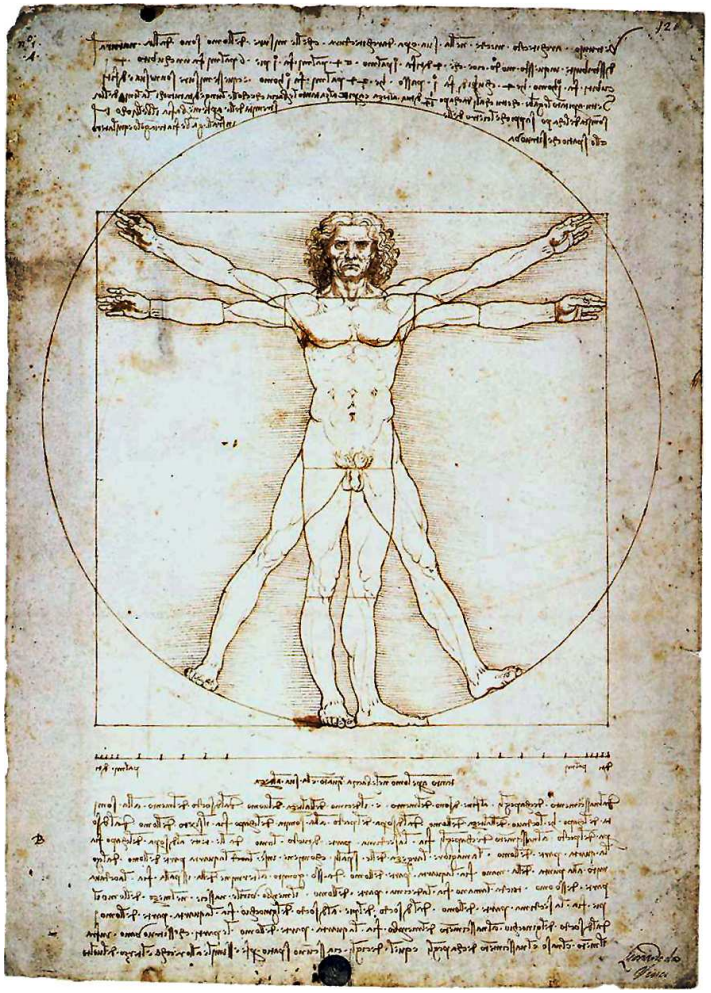
Contents

1 Software architecture

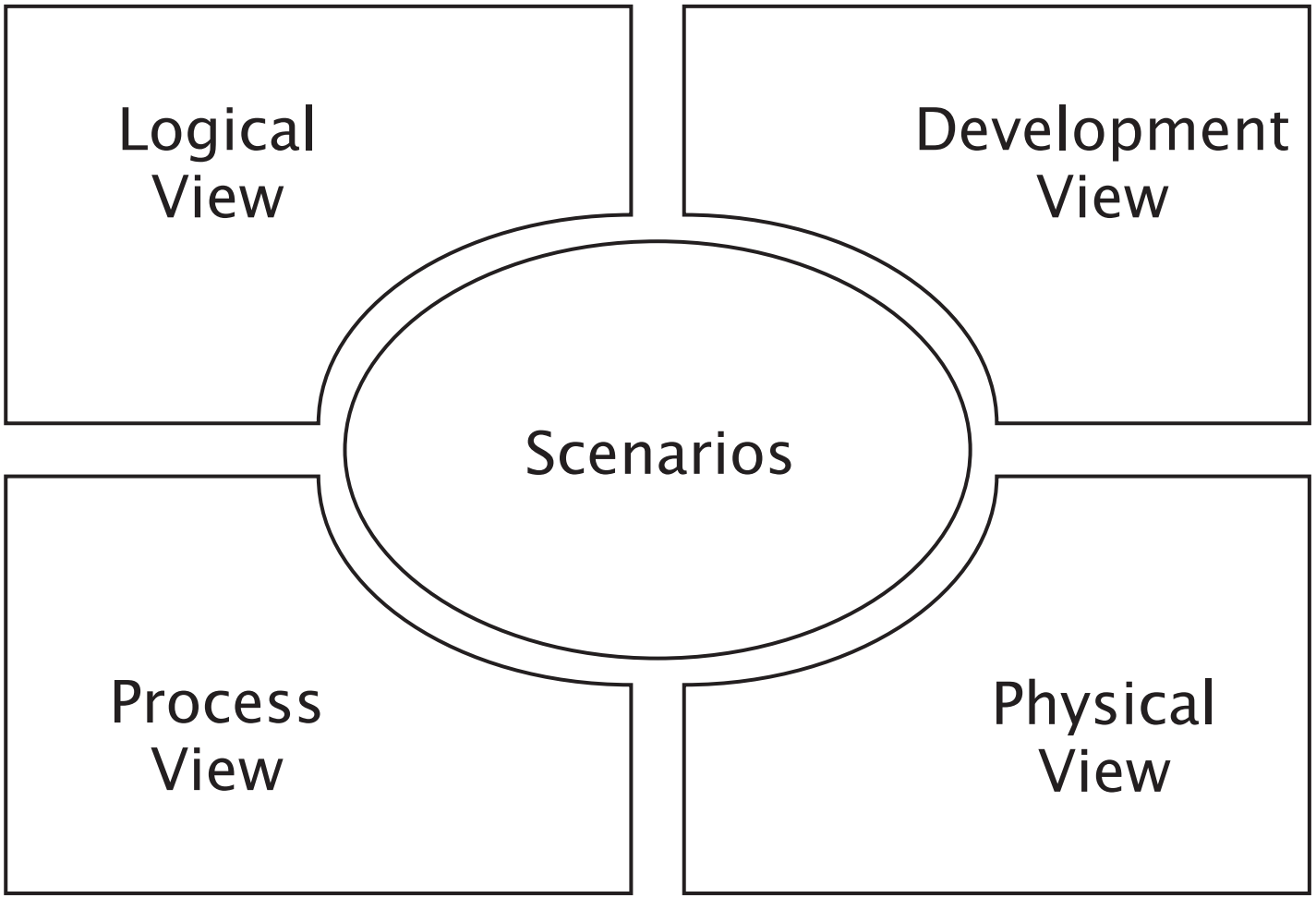
- design beyond the level of algorithms and data structures
- as abstract datatypes themselves are design beyond the level of individual statements
- gross organization; global control structure; protocols for communication, synchronization and data access; physical distribution...
- Garlan and Shaw, 1994, and much subsequent work
  eg architectural *patterns*
- orthogonal to underlying technology: WS, CORBA...

1.1 Vitruvian Triad: firmitas, utilitas, venustas

- Vitruvius: Roman, 1st century BC
- *Ten Books on Architecture*
- function (utilitas, ‘utility’)
- form (venustas, ‘beauty’)
- fabrication (firmitas, ‘soundness’)
- architectural principles
- retrospective
- perhaps a bit idealist

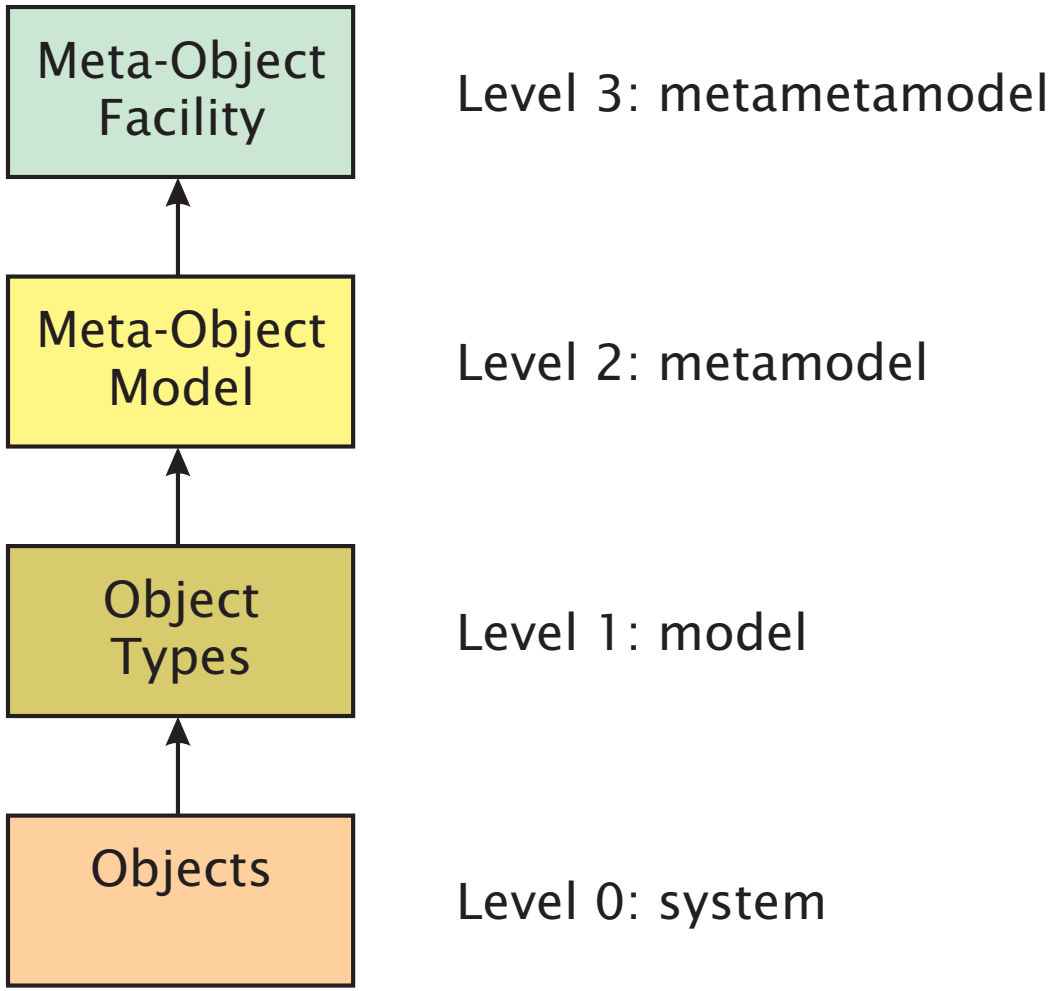


1.2 The 4+1 View Model of Architecture

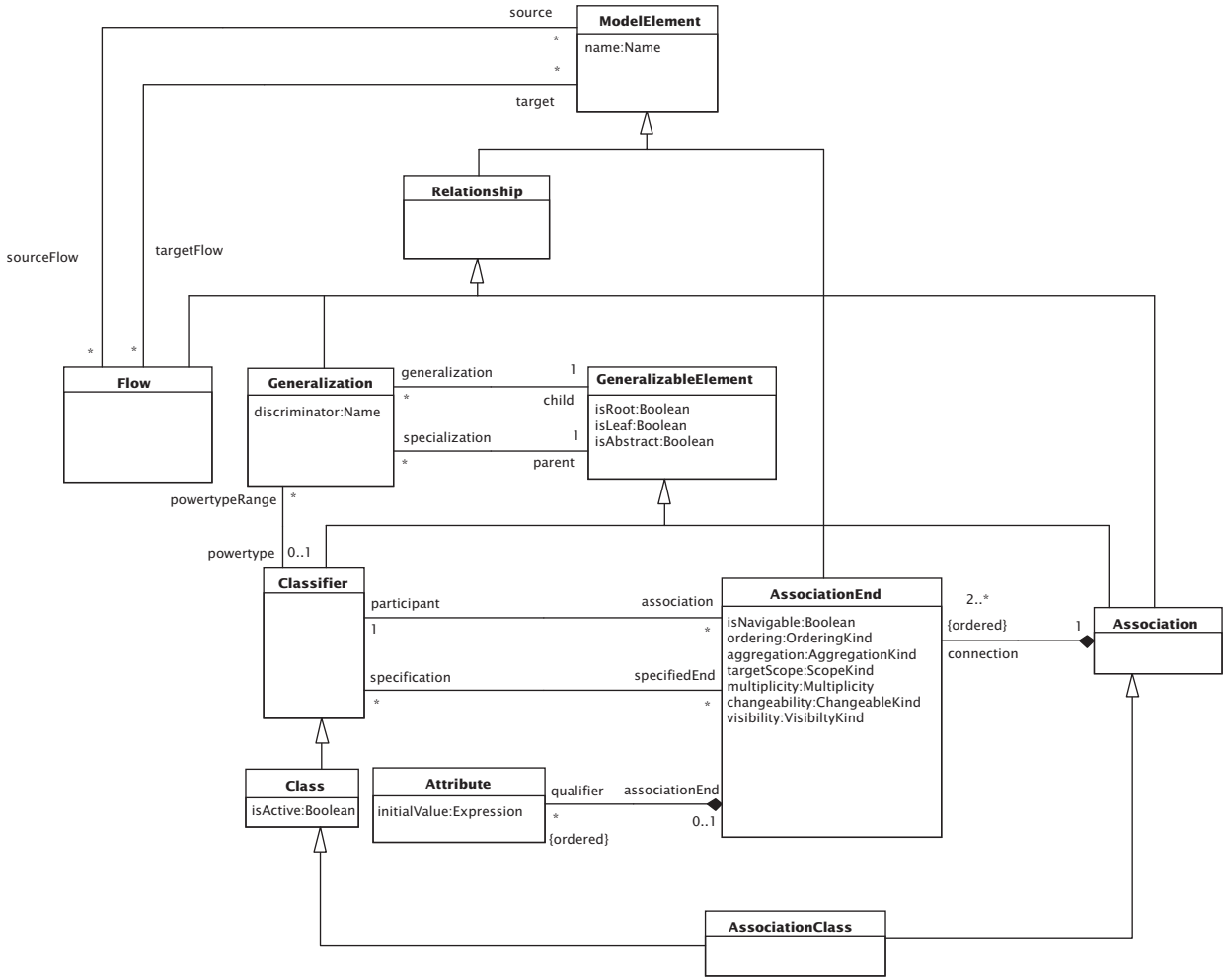


(Philippe Kruchten, *IEEE Software* 12(6) 1995)

1.3 Models and metamodels



1.4 UML metamodel (part)



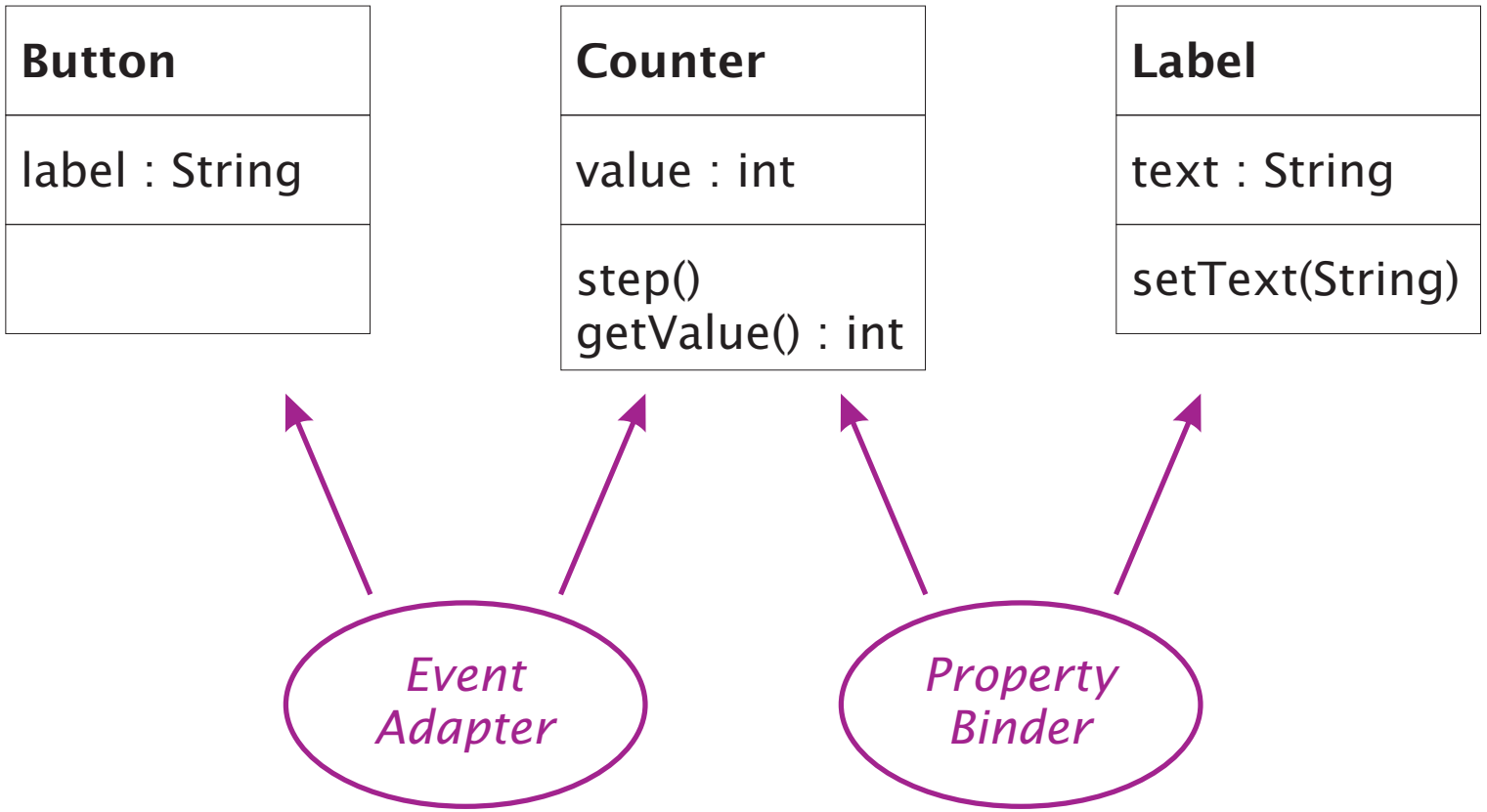
1.5 Components and connectors

- *components*: sequential, single-threaded *computations*
- *connectors*: concurrent, multi-threaded *coordination*
- connectors should be first-class entities (cf workflow)
- paradigm of system assembly by *superimposition*
- communication, coordination, monitoring, control

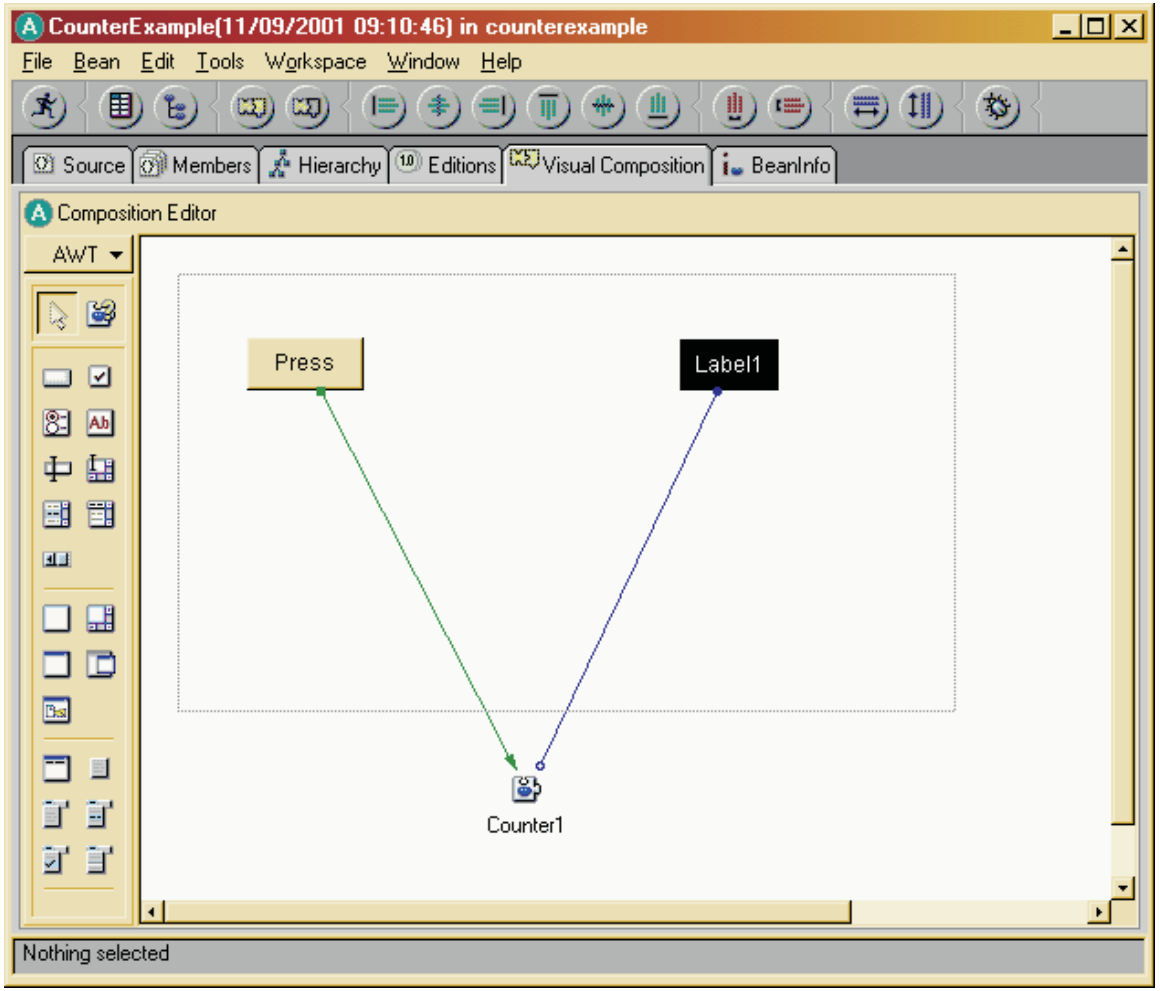
Counter-example

User presses button; counter steps; label displays counter value.

Connectors for coordination



Visual programming



1.6 Product line architectures

- “set of programs studied in terms of the common and special properties of the individual members” (Parnas)
- family of related but different products
- separate the generic from the specific
- eg mobile phones
- eg engine management

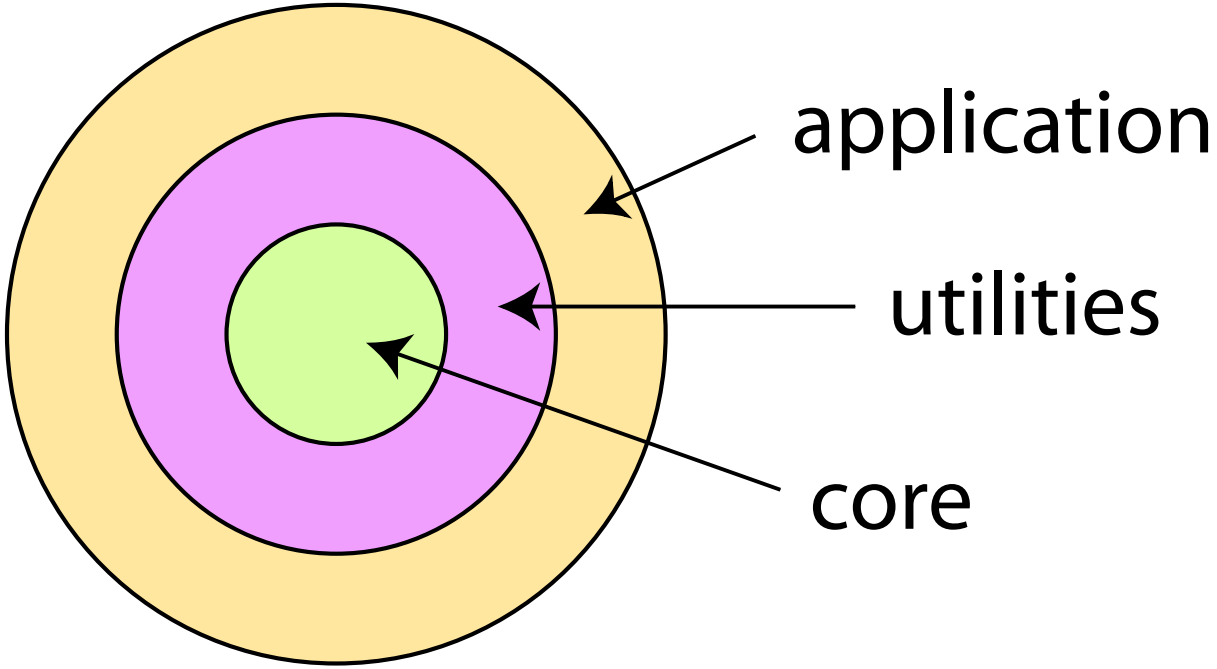
1.7 BMW car configurator

(idea from Roberto Lopez on SPL)

2 Client–server

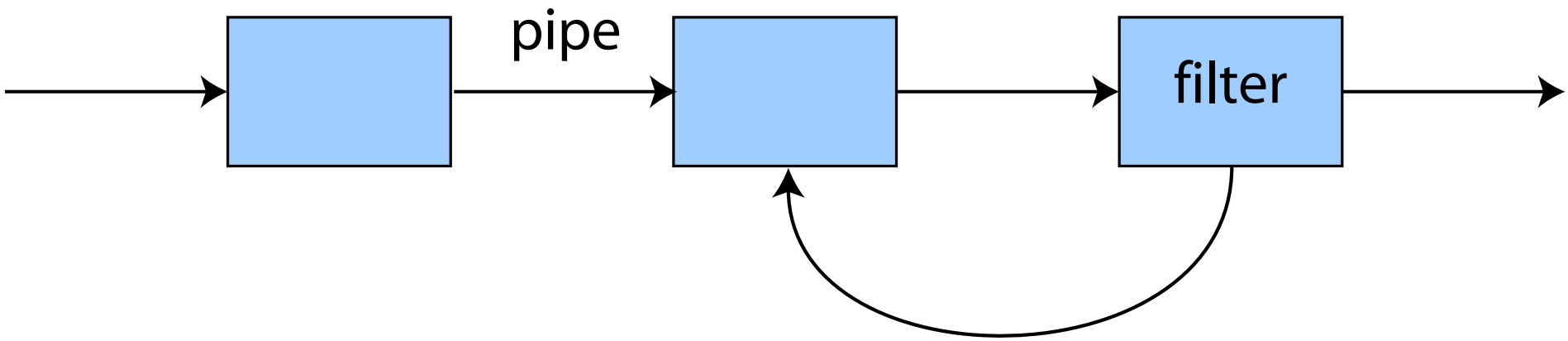
- most common and familiar architecture for distributed systems
- *server* offers a set of services, and waits for service requests
- *client* sends request to server, awaits response
- client is triggering process, initiates synchronization, may be finite; server is reactive, typically runs indefinitely
- multi-tier client–server architectures (eg with client-side proxies and server-side gateways) are *layered*...

3 Layered



- hierarchical organization
- each layer provides a *service* to the layer above, and is a *client* of the layer below
- strictly, only adjacent layers interact (although a layer may import certain services and re-export them); each layer implements a *virtual machine*
- loosely, upper layer may access any lower layer; then layers are translucent rather than opaque
- eg layered communications protocols: OSI seven-layer model
- challenges: often efficiency considerations encourage breaking abstraction boundaries

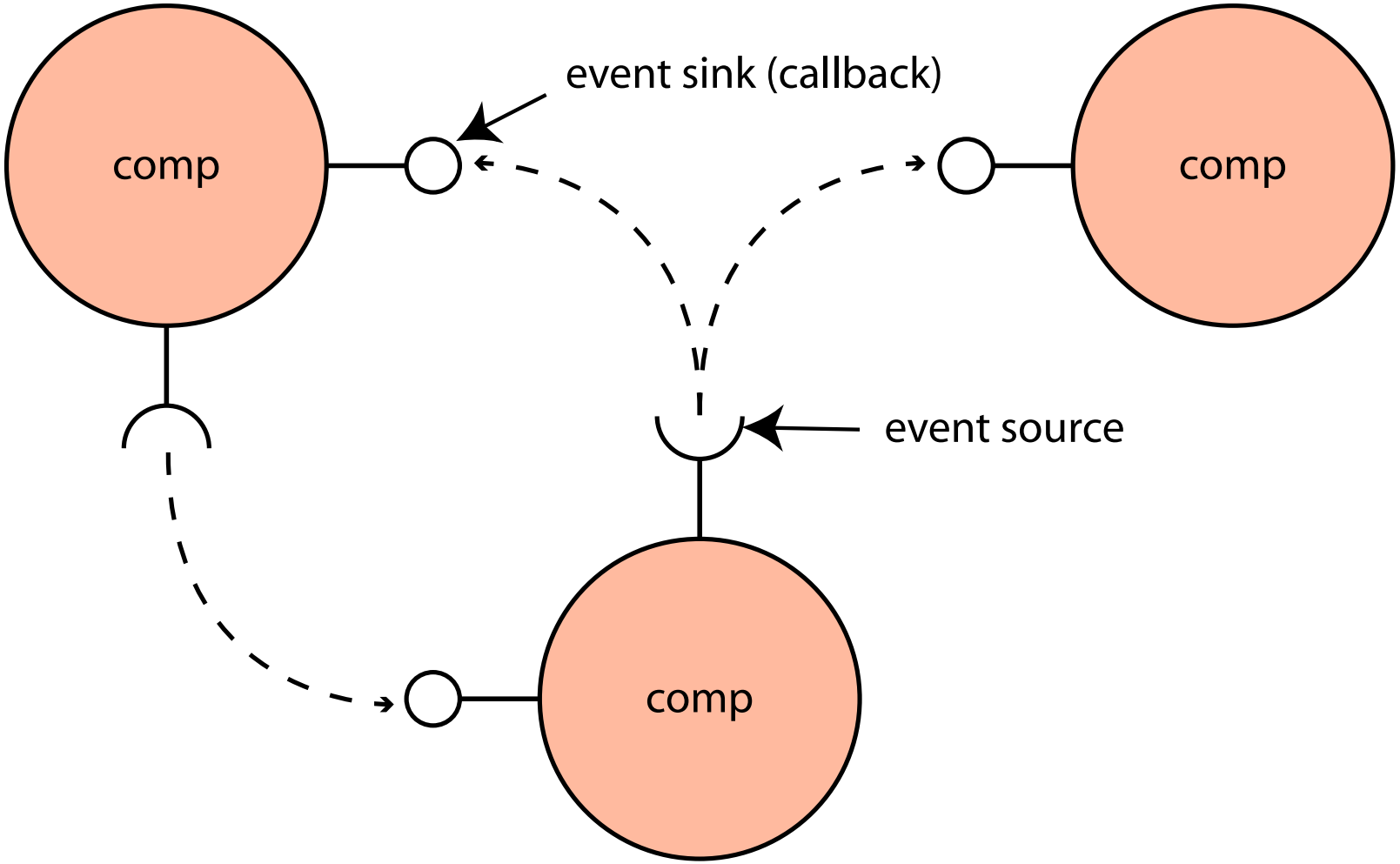
4 Pipes and filters



- *filter* reads a stream of data from input(s), writes a stream of data to output(s)
- *pipe* connects output of one filter to input of another

- filter typically applies a local transformation to the data
- execution usually incremental, on demand: filter may start writing before finishing reading, so filters may compute in parallel
- filters are independent: no shared state, no dependence on characteristics of upstream or downstream neighbours
- specializations: linear *pipelines*, *bounded* buffering, *typed* data, degenerate *batched sequential*
- eg Unix processes; compiler phases; signal processing; systolic arrays

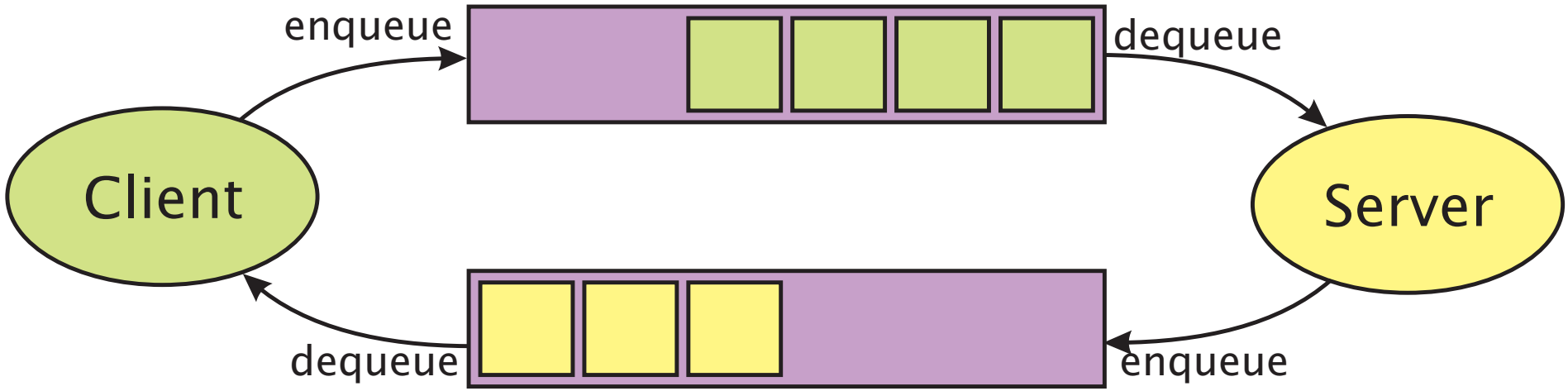
5 Event-based, implicit invocation



- instead of explicitly invoking a (perhaps remote) procedure, component triggers an *event*
- other components register interest in such events, perhaps by specifying a *callback*
- when event is triggered, *event manager* (or perhaps the triggering component itself) invokes all appropriate registered callbacks
- event source needs no knowledge of event sinks
- challenges: non-determinacy, asynchrony, passing data
- eg database triggers, user interfaces, the OBSERVER pattern, 'pub–sub', EDA, CEP

5.1 Message-oriented

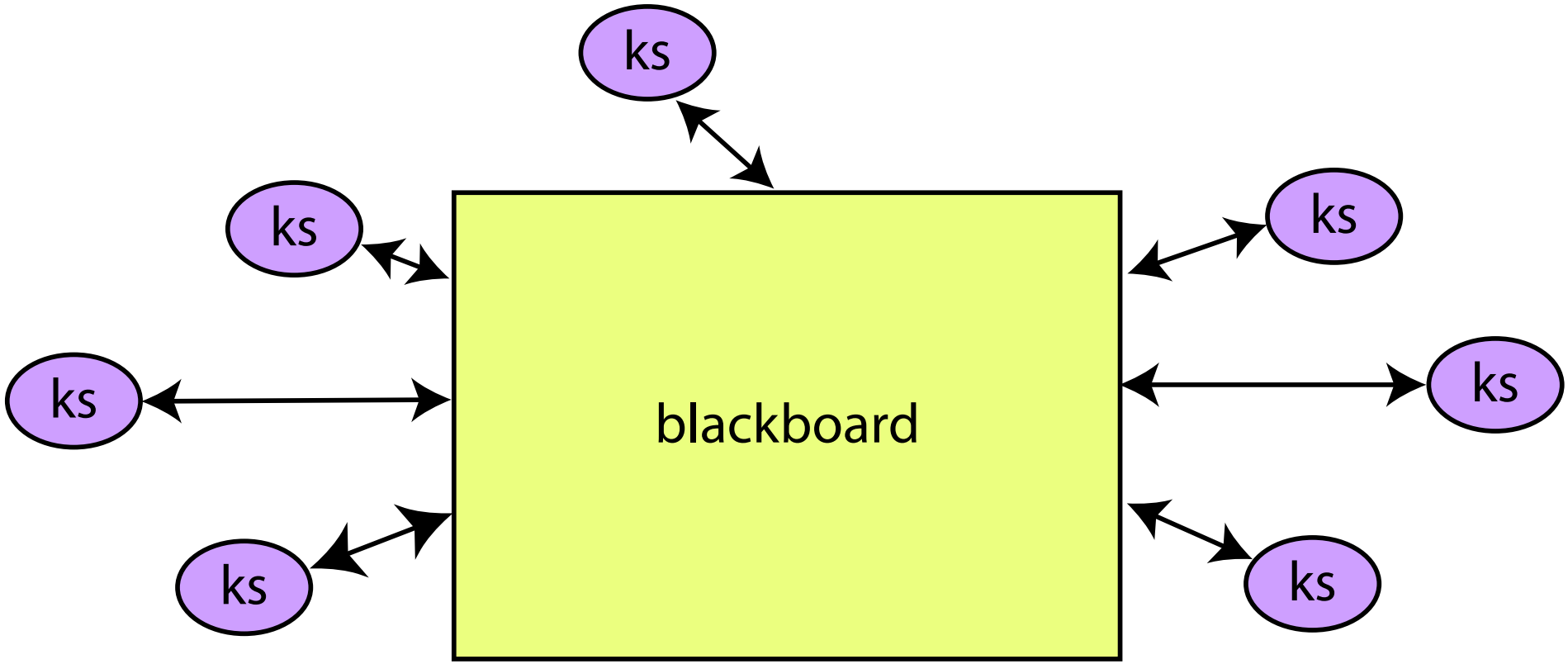
- decouples components by interposing *message queues*
- induces *asynchronous communication*: components need not be simultaneously connected
- message consumers often notified by events
- may require (and typically provides) persistent storage for queued messages, improving reliability
- eg MQSeries, MSMQ, JMS, WS-ReliableMessageing



5.2 Complex event processing

- inference of higher-level events from lower-level
- eg (bells + tux + gown + rice) nearby ⇒ wedding
- abstraction, correlation, causality, timing
- algorithmic trading, fraud detection, embedded systems
- see `www.complexevents.com`

6 Repository



- central *store* maintains global state
- independent *knowledge sources* interact with central store
- KS responds opportunistically when stored state makes it applicable
- (alternatively, actions may be triggered by input types; then more like a database with stored procedures)
- eg Blackboard systems (HEARSAY-II speech recognizer)

6.1 Tuple spaces

- a style of *coordination language*: separate communication from computation
- tuple space is *bag* of tagged data
- tagged data is added, examined and removed by concurrent processes (with locking)
- tuples may represent *tasks* and *results*
- eg *Linda*, *Chemical Abstract Machine*, *JavaSpaces*, *Space-Based Architecture*...

7 Peer-to-peer

- clients and servers coincide: *servents*
- typically an expectation that users of services will reciprocate
- minimize centralization, eg of registry, broker or router
- how do servents locate each other? perhaps initial use of registry, then boot-strapping ('hybrid p2p')
- reduces bottlenecks, increases robustness
- *unstructured* vs *structured* overlay networks: arbitrary or deterministic allocation of resources?
- eg file-sharing architectures such as Napster, Gnutella, BitTorrent

7.1 Agent-based

- autonomous components: no central control
- reasoning about environment and interactions
- belief vs knowledge: modal and epistemic logic
- devolved responsibility: action on behalf of user
- maybe mobile

8 Interpreter

- virtual machine in software
- pseudo-program *interpreted*
- multiple levels of ‘state’: interpreting and interpreted
- eg *rule-based system*
  - fixed body of *rules*
  - varying knowledge base of *facts*
  - interpretation of rules by *inference engine*

9 Grid/Cloud/Ubicomp...

- massively parallel, loosely coupled
- ready access to shared resources
  (processor, space, equipment, expertise...)
- pay-per-use, on-demand
- "autonomic" (self-managed)

9.1 Grid computing

- by analogy with electricity grid
- pioneered by SETI@Home

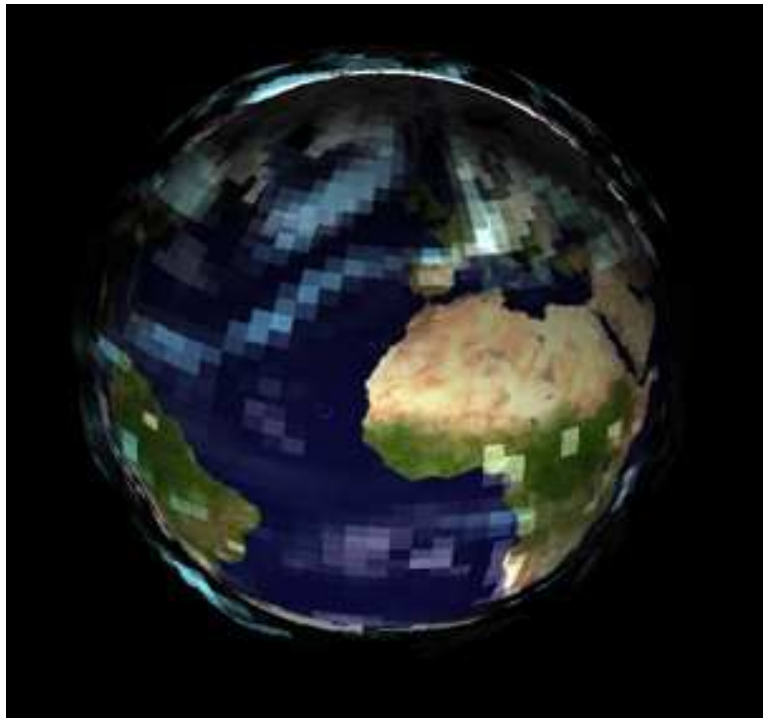
  3 million volunteers downloading and analyzing radio telescope data
- loosely coupled, heterogeneous, geographically dispersed
- *virtual organizations*
- move computation to data rather than vice versa (eg LHC)

9.2 Grid services

- web services for grid computing
- stateful, by necessity (now exploiting WSRF)
- FACTORY pattern for service instantiation
- see Open Grid Services Architecture (OGSA)
  `http://www.globus.org/alliance/publications/papers/ogsa.pdf`

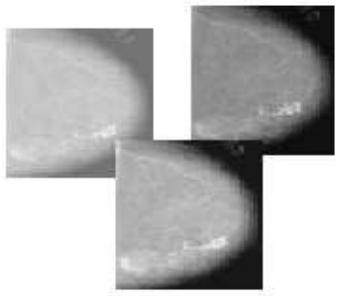
9.3 climate*prediction*.net

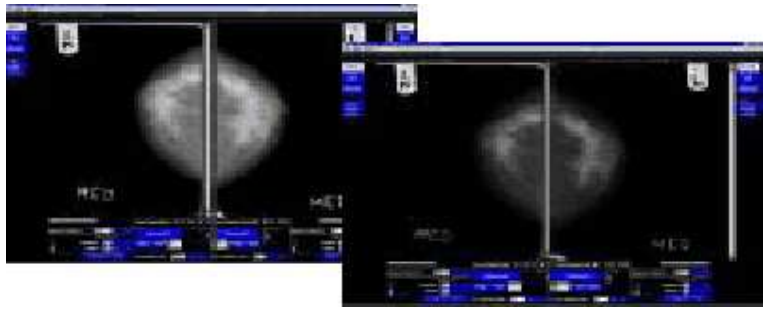
- users donate computer time
- in the style of SETI@home
- but jobs run for weeks, not hours
- running a climate model from the Met. Office
- upload 10Mb of data to servers around the world
- build probabilistic model to predict where we'll be in 2050



9.4 eDiamond

- national database of mammographic images
- applications in training and diagnosis
- Oxford, IBM, various hospitals
- 12MB per image . . .
- major requirements in security, data location, access times





9.5 Cloud computing

- “software as a service”, rented not bought
- services and data hosted online, available anywhere
- data centres, virtualization technologies
- ‘The Cloud’ is a metaphor for the internet (eg in network diagrams)
- eg Amazon WS, Google Apps, Microsoft Azure

9.6 Map-reduce

- Google's programming model for cluster computing
- inspired by functional programming
- works on sets of key/value pairs
- *map* over each pair a function giving intermediate k/v pairs
- for each key, *reduce* all associated values to some summary
- framework handles execution on commodity hardware: partitioning, scheduling, communication, fault-tolerance
- many examples: histograms (wordcount, access stats), distributed grep and sort, reverse web links, inverted index...
- open-source implementation as Apache Hadoop

9.7 Ubiquitous computing

- information processing in everyday objects
- handheld devices, consumer goods, sensors, RFID...
- ad-hoc networks, innovative UIs
- “pervasive computing”, “Internet of Things”, “everyware”
- see MOB

Index

Contents

Service-Oriented Architecture

Monday	Tuesday	Wednesday	Thursday	Friday
Introduction Components	REST	Composition	Architecture	Engineering
coffee	coffee	coffee	coffee	coffee
Components	REST	Composition	Architecture	Conclusion
lunch	lunch	lunch	lunch	lunch
Web Services	Qualities	Objects	Semantic Web	
tea	tea	tea	tea	
Web Services	Qualities	Objects	Semantic Web	