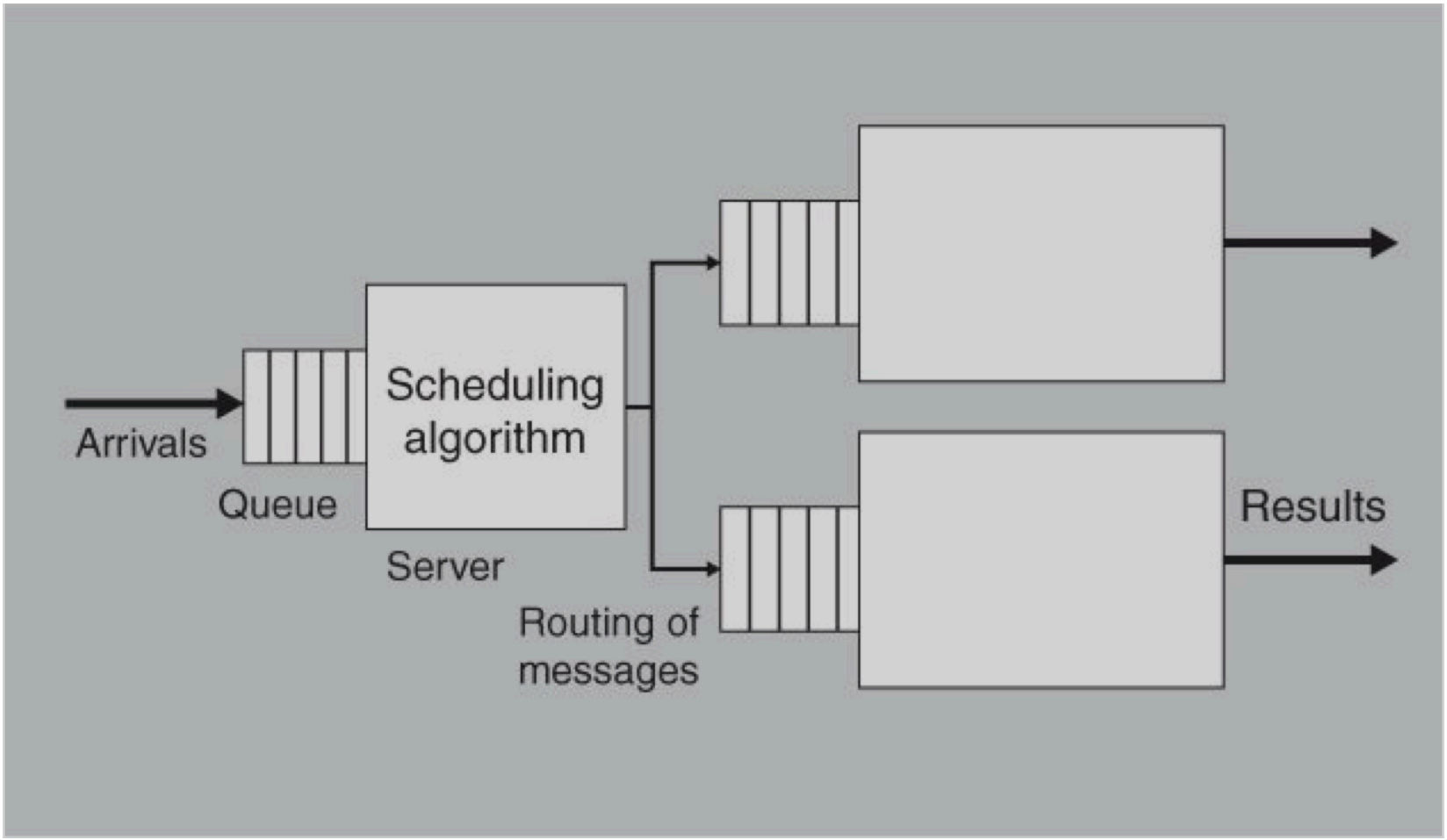
Architectural Modelling

Overview

- Software Architecture is intended to give us control over Quality Attributes.
- Ideally we'd like to be able to use Software Architecture to predict Quality Attributes.
- We should be able to build a predictive model of the Software Architecture and use the model to predict Qas.
- The current situation is patchy…

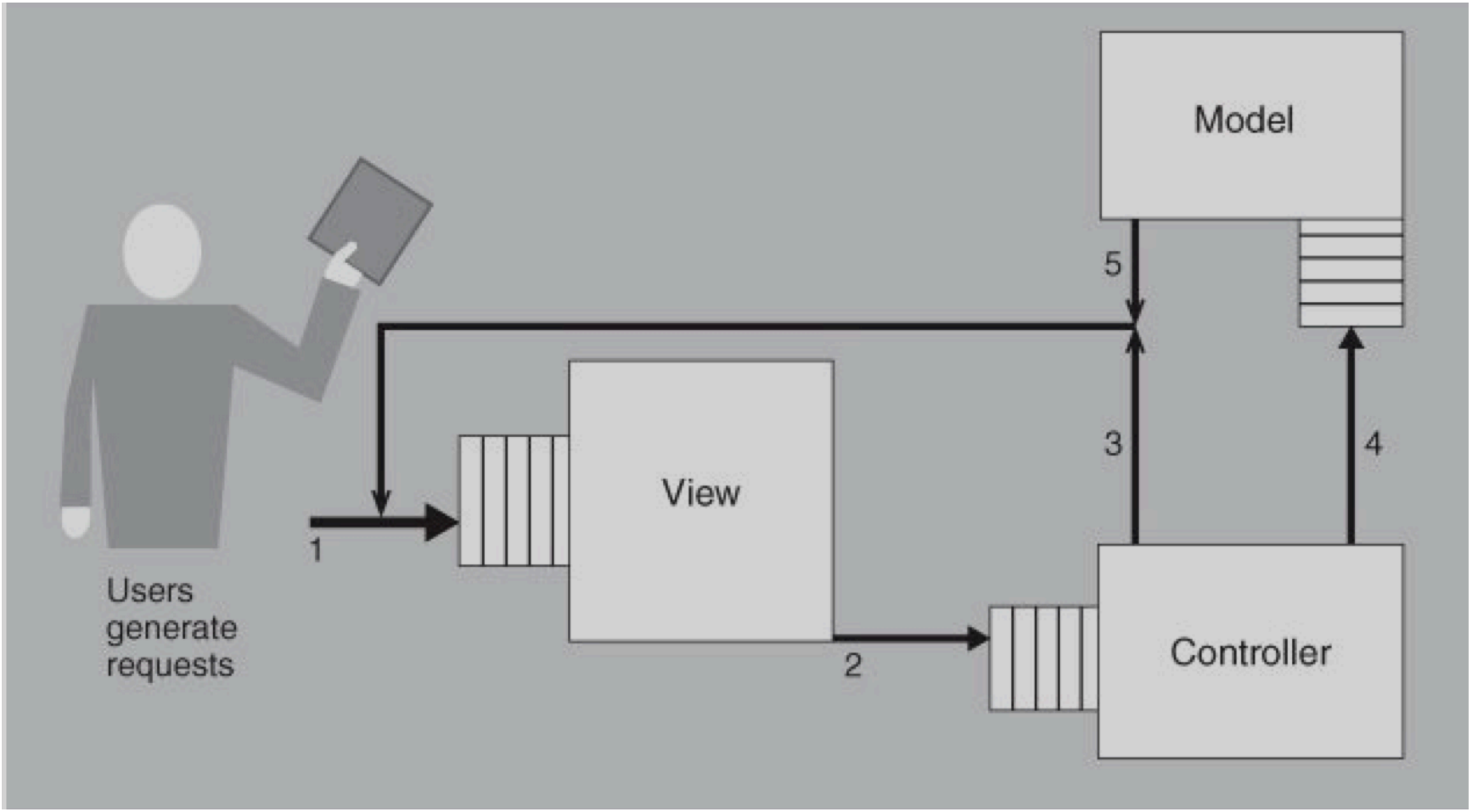
Performance: Queueing Models



Data Needed for the Model

- We need the following information in order to model effectively:
  - The distribution for the arrival of service requests
  - The queuing discipline
  - The scheduling algorithm
  - The distribution of service times for service requests
  - Network characteristics
- The theory places restrictions on the distributions
  - Arrivals are usually expected to be Poisson Distributions specified by arrival rate
  - Service times are usually exponentially distributed on the service rate.
  - Some queuing behaviors are excluded such as reneging or jockying

Queueing Model of MVC



Data for MVC

1. Rate of service requests: the View component will service them at some rate.
2. Service requests to the Controller are generated by the View component.
3. Service requests from the Controller to the View component
4. Service requests from the Controller to the model
5. Service requests from the Model to the View Component

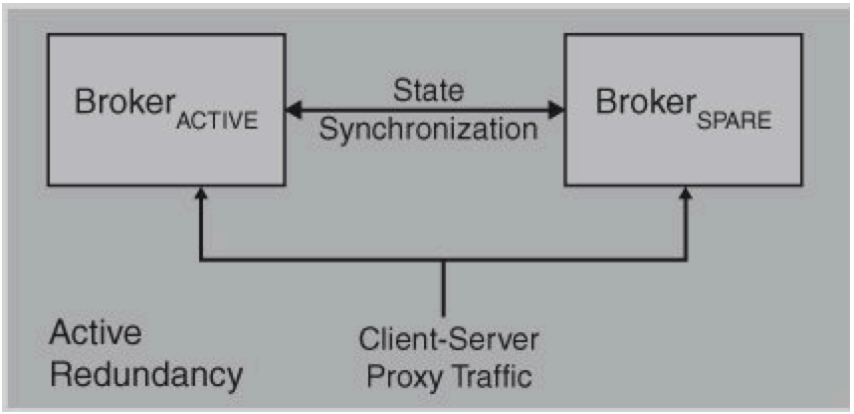
Modelling MVC

- We need estimates of:
  - Distribution of external service demands
  - Queuing Disciplines within the queues in front of each component.
  - Network latencies
  - Transfer characteristics:
    - View – Controller
    - Controller – View
    - Controller – Model
    - Model – View
- This is a well-studied area and can produce good predictions if the estimates are good.
- More sophisticated techniques are available.
- Scaling to large numbers of components is an issue

Availability

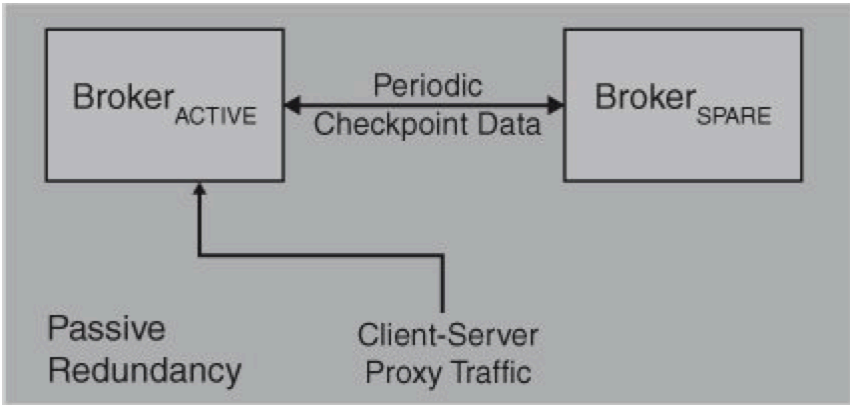
- Recall availability is defined as:
  - *mtbf/(mtbf+mttr)*
- Not as well developed as Performance
- Range of different architectural solutions result in different availabilities.
- One key issue is how long it takes to detect that a failure has taken place…
- Example is a Broker system.

Hot Spare



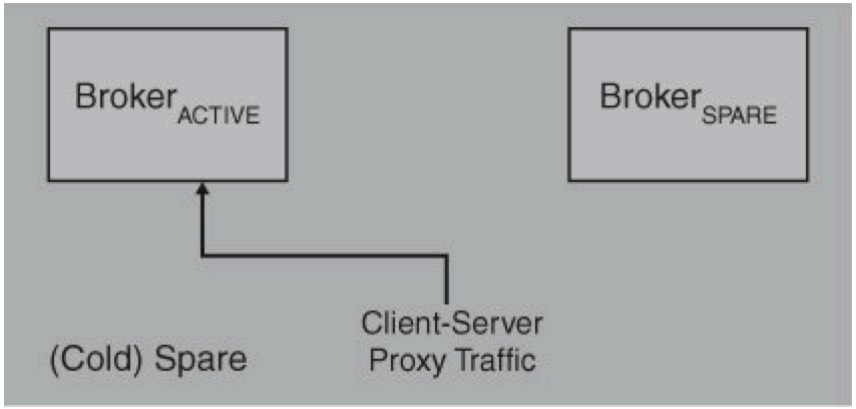
- Active and redundant both receive identical request stream.
- Synchronous maintenance of broker state.
- Fast failover in the event of failure of the active system.

Warm Spare



- Warm broker is maintained at the most recent checkpoint state.
- In the event of failure the system rolls back to the most recent checkpoint.
- This is slower than the hot spare approach

Cold Spare



- Here there is no attempt to synchronise.
- In the event of failure the cold spare is started.
- The system state is recovered via interaction with other systems (so they have to be resilient to failure in the broker)

Analytical Model Landscape

Quality Attribute	Intellectual Basis	Maturity/Gaps
Availability	Markov models; statistical models	Moderate maturity; mature in the hardware reliability domain, less mature in the software domain. Requires models that speak to state recovery and for which failure percentages can be attributed to software.
Interoperability	Conceptual framework	Low maturity; models require substantial human interpretation and input.
Modifiability	Coupling and cohesion metrics; cost models	Substantial research in academia; still requires more empirical support in real-world environments.
Performance	Queuing theory; real-time scheduling theory	High maturity; requires considerable education and training to use properly.
Security	No architectural models	
Testability	Component interaction metrics	Low maturity; little empirical validation.
Usability	No architectural models	

Analysis in the Lifecycle

Life-Cycle Stage	Form of Analysis	Cost	Confidence
Requirements	Experience-based analogy	Low	Low–High
Requirements	Back-of-the-envelope	Low	Low–Medium
Architecture	Thought experiment	Low	Low–Medium
Architecture	Checklist	Low	Medium
Architecture	Analytic model	Low–Medium	Medium
Architecture	Simulation	Medium	Medium
Architecture	Prototype	Medium	Medium–High
Implementation	Experiment	Medium–High	Medium–High
Fielded System	Instrumentation	Medium–High	High

Types of Analysis

- Thought experiment: just a sort of discussion using informed people.
- Back of the envelope: using very approximate techniques with unreliable assumptions.
- Checklist: collated experience.
- Analytic Model: based on sound abstractions –heavily dependent on estimates being correct
- Simulation: higher level of detail – less analytic, more concrete.
- Prototype: approximate system in an experimental setup.
- Experiment: fielded system, simulated load
- Instrumentation: measuring the variable of interest.

Summary

- Architecture is the correct level to deal with Quality Attributes.
- Some Qas have good, well established, analysis techniques, others don’t.
- Analysis can be costly depending on how accurate you want it to be.
- Analysis through the lifecycle helps with decision taking.