TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Programming Distributed Systems

12 Programming Models for Distributed Systems

Annette Bieniusa

AG Softech
FB Informatik
TU Kaiserslautern

Summer Term 2018

What is a Programming Model?[3]

- A programming model is some form of abstract machine
  - Provides operations to the level above
  - Requires implementations for these operations on the level(s) below
- Simplification through abstraction
- Standard interface that remains stable even if underlying architecture changes
- Provide different levels of abstraction
- Often starting point for language development

$\Rightarrow$ Separation of concern between software developers and framework implementors (runtime system, compiler, etc.)

Properties of good programming models

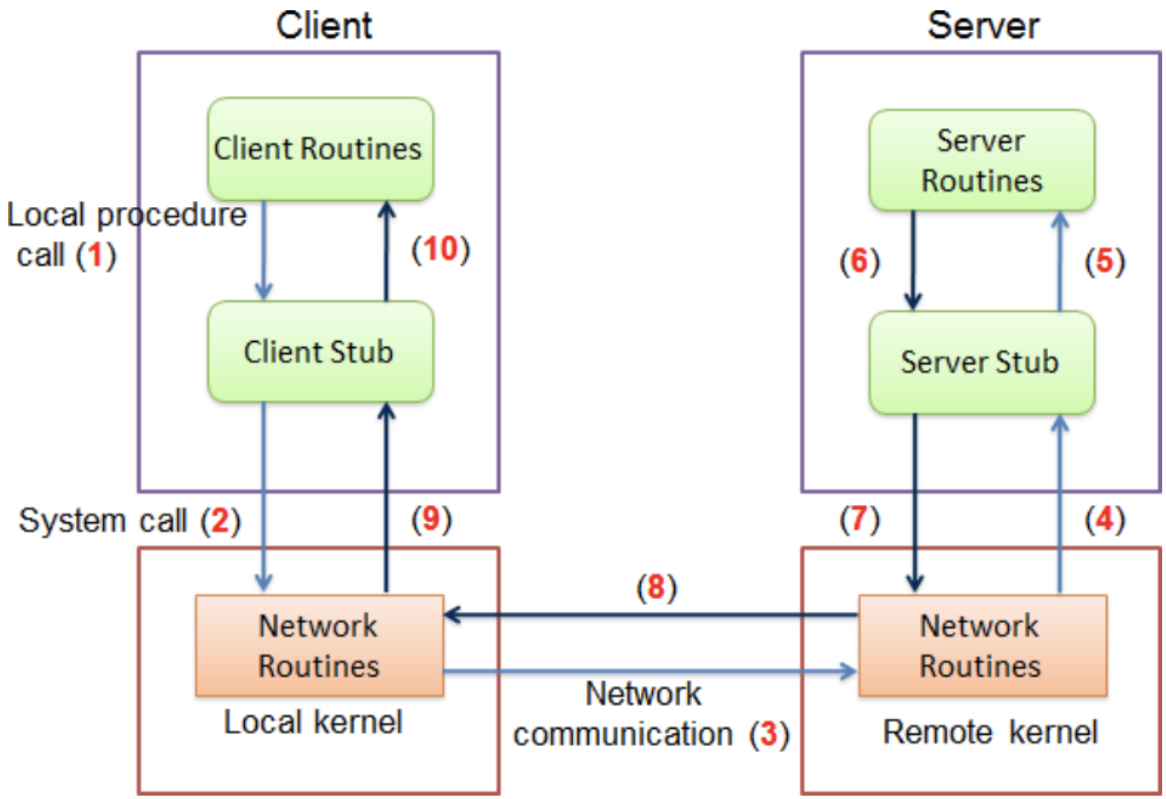
- Meaningful abstractions
- System-architecture independent
- Efficiently implementable
- Easy to understand

What kind of abstractions should a programming model for distributed systems provide?

Remote Procedure Call

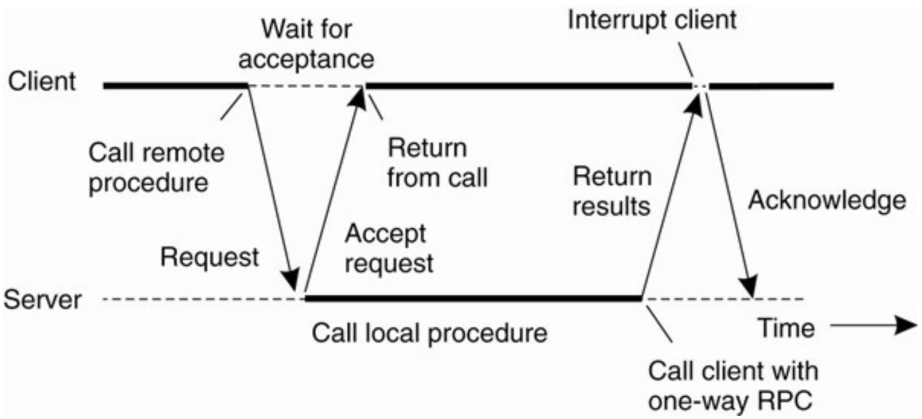
Remote Procedure Call (RPC)

- Rather broad classifying term with changing meaning over time
  - From client-server design to interconnected services
- *Two entities (caller/callee) with different address spaces communicate over some channel in a request-response mechanism*
- Examples: CORBA (Common Object Request Broker Architecture), Java RMI (Remote Method Invocation), SOAP (Simple Object Access Protocol), gRPC, Finagle ...


Client
Server
Client Routines
Server Routines
Local procedure call (1)
(10)
(6)
(5)
Client Stub
Server Stub
System call (2)
(9)
(7)
(4)
(8)
Network Routines
Network Routines
Network communication (3)
Local kernel
Remote kernel

Aspects of modern RPC

- Language-agnostic
- Serialization (aka marshalling or pickling)
  - JSON, XML, Protocol Buffers, ...
- Load-balancing
  - Microservice architectures!
- Asynchronous



$\Rightarrow$ RPC as term gets more and more diffuse

Futures and Promises

- "Asynchronous RPC"
- A future is a value that will eventually become available
- Two states:
  - *completed*: value is available
  - *incomplete*: computation for value is not yet complete
- Strategies: Eager vs. lazy evaluation
- Typical application: Web development and user interfaces

Actors and Message Passing

Characteristics of Actor Model[2]

- Actors are isolated units of computation+state that can send messages asynchronously to each other
- Messages are queued in mailbox and processed sequentially when they match against some pattern/rule
- No assumptions on message delivery guarantees
- (Potential) State+behavior changes upon message processing[1]
- Very close to Alan Kay's definition of Object-Oriented Programming

Actors in the Wild

- Erlang
  - Process-based
  - Pure message passing
  - `monitor` and `link` for notification of process failure/shutdown
  - OTP (Open Telecom Platform) for generic reusable patterns
- Akka
  - Actor model for the JVM
  - Purges non-matching messages
  - Enforces parental supervision
  - Included in Scala standard library
- Orleans
  - Cloud computing
  - Scalability by replication
  - Fine-grain reconciliation of state with transactions

And more programming models

- "Big data"
- Batch-processing
  - Static data sets allow to distribute computation
  - Typically large latencies
- Stream-processing
  - (Infinite) Sequence of data being pushed to processors
  - Producers vs. Consumers
  - Notion of windows and time
  - Example: Kafka

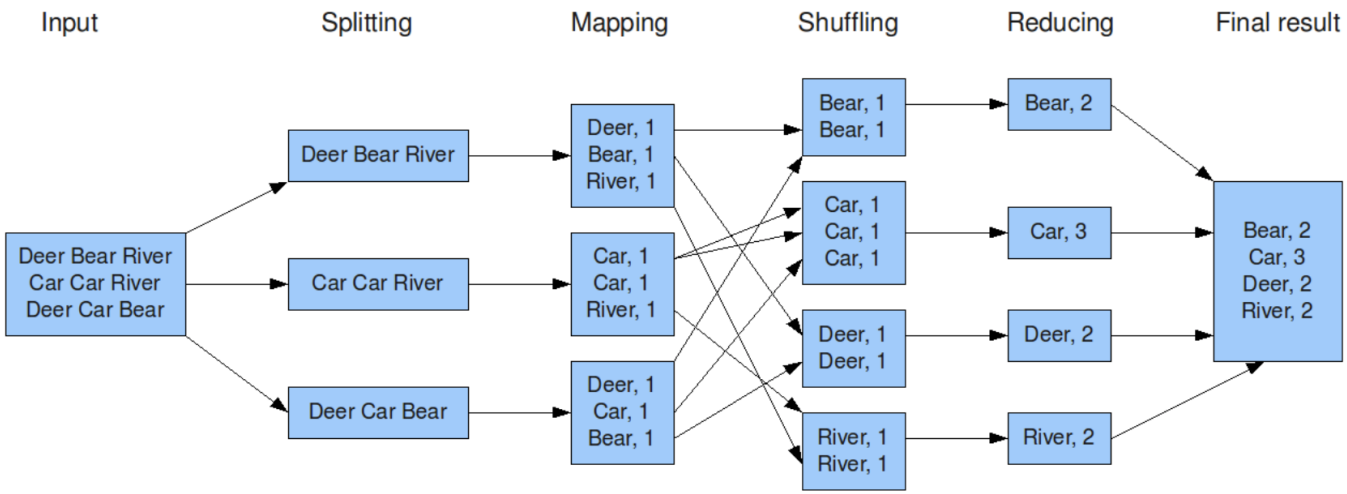
The Future: Distributed Programming Languages

From Model to Language

- Challenges: Partial failure, concurrency and consistency, latency, ...

1. Distributed Shared Memory
   - Runtime maps virtual addresses to physical ones
   - "Single-system" illusion
2. Actors
   - Explicit communication
   - Location of processes is transparent
3. Dataflow
   - Data transformations expressed as DAG
   - Processes are transparent
   - Example: MapReduce (Google), Dryad (Microsoft), Spark

Example: WordCount in MapReduce



Further reading

- Material collection by Northeastern University, CS7680 Special Topics in Computing Systems: Programming Models for Distributed Computing

[1] Gul Agha. "Concurrent Object-Oriented Programming". In: *Commun. ACM* 33.9 (1990), S. 125–141. DOI: 10.1145/83880.84528. URL: http://doi.acm.org/10.1145/83880.84528.

[2] Carl Hewitt, Peter Boehler Bishop und Richard Steiger. "A Universal Modular ACTOR Formalism for Artificial Intelligence". In: *Proceedings of the 3rd International Joint Conference on Artificial Intelligence. Standford, CA, USA, August 20-23, 1973*. Hrsg. von Nils J. Nilsson. William Kaufmann, 1973, S. 235–245. URL: http://ijcai.org/Proceedings/73/Papers/027B.pdf.

[3] David B. Skillicorn und Domenico Talia. "Models and Languages for Parallel Computation". In: *ACM Comput. Surv.* 30.2 (1998), S. 123–169. DOI: 10.1145/280277.280278. URL: http://doi.acm.org/10.1145/280277.280278.