

Concurrency models, actors and Erlang

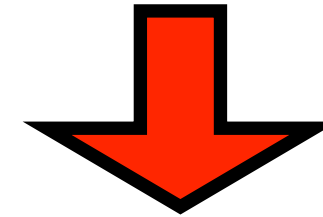
Module 8

Lecture

Operating systems and process oriented programming 2020

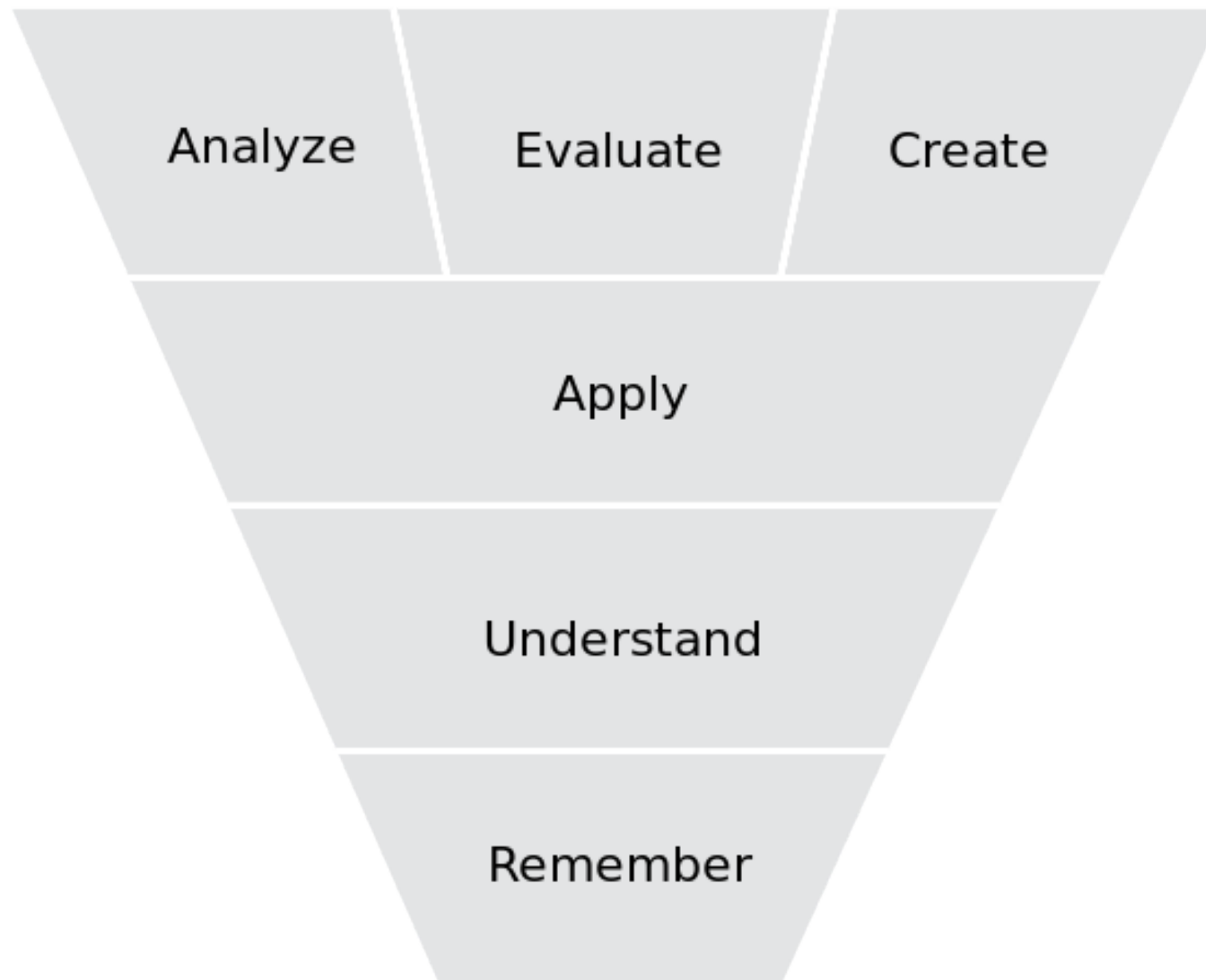
1DT096

Course timeline



Period 4				Modul 7			Modul 8		Modul 9	
W	D	Date	Bivillkor	OS	OSPP	DSP	OSPP	DSP	OSPP	DSP
13	Mo	2020-03-23					L			
	Tu	2020-03-24				W	L + T			
	We	2020-03-25					W + T			
	Th	2020-03-26				P	T		L	
	Fr	2020-03-27					S			
14	Mo	2020-03-30				Pr (LÅ)	C		W + D	
	Tu	2020-03-31							W	Wp - h
	We	2020-04-01							P	Wp - h
	Th	2020-04-02					Tr		D	Pp - h
	Fr	2020-04-03							W	H
	Mo	2020-04-06					Sr+Cr+Ch			

The cognitive domain

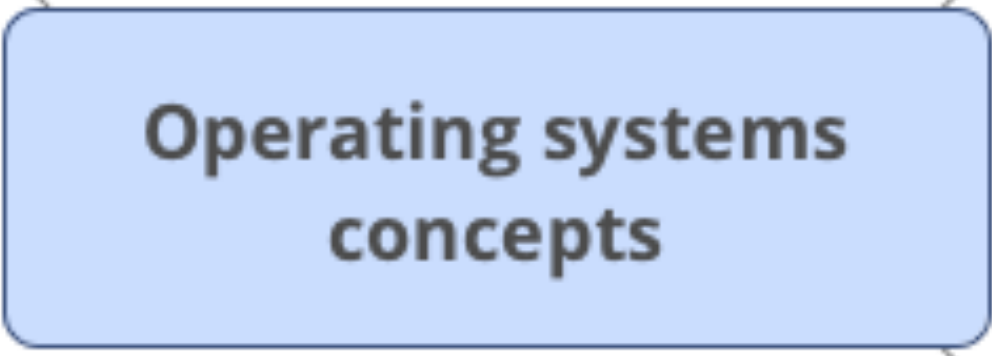


**Operating
system
concepts**

Operating system concepts

(1)

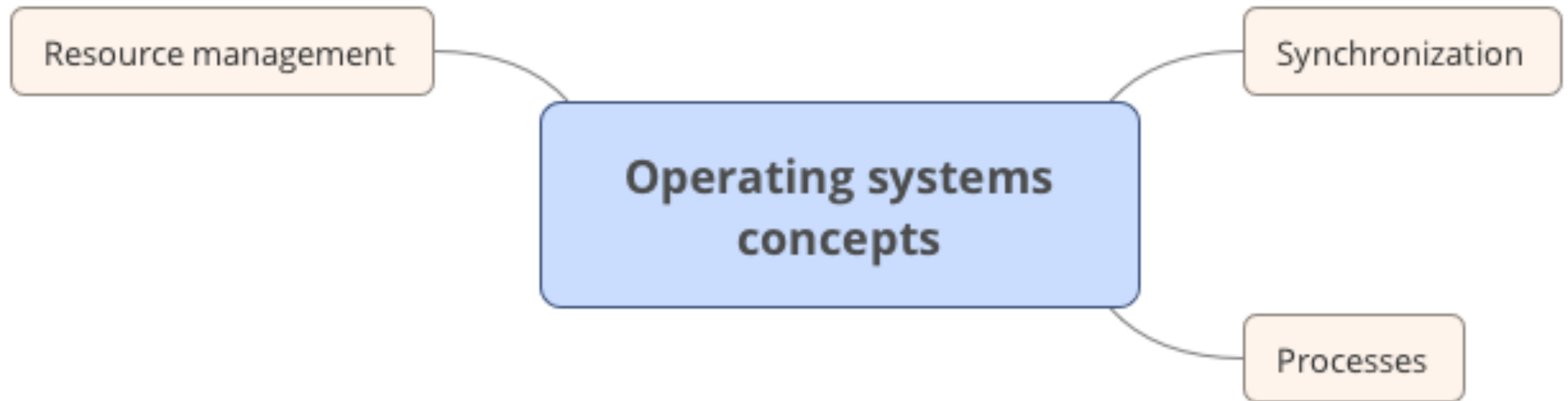
Important concepts and how they relate to each other.



Operating systems
concepts

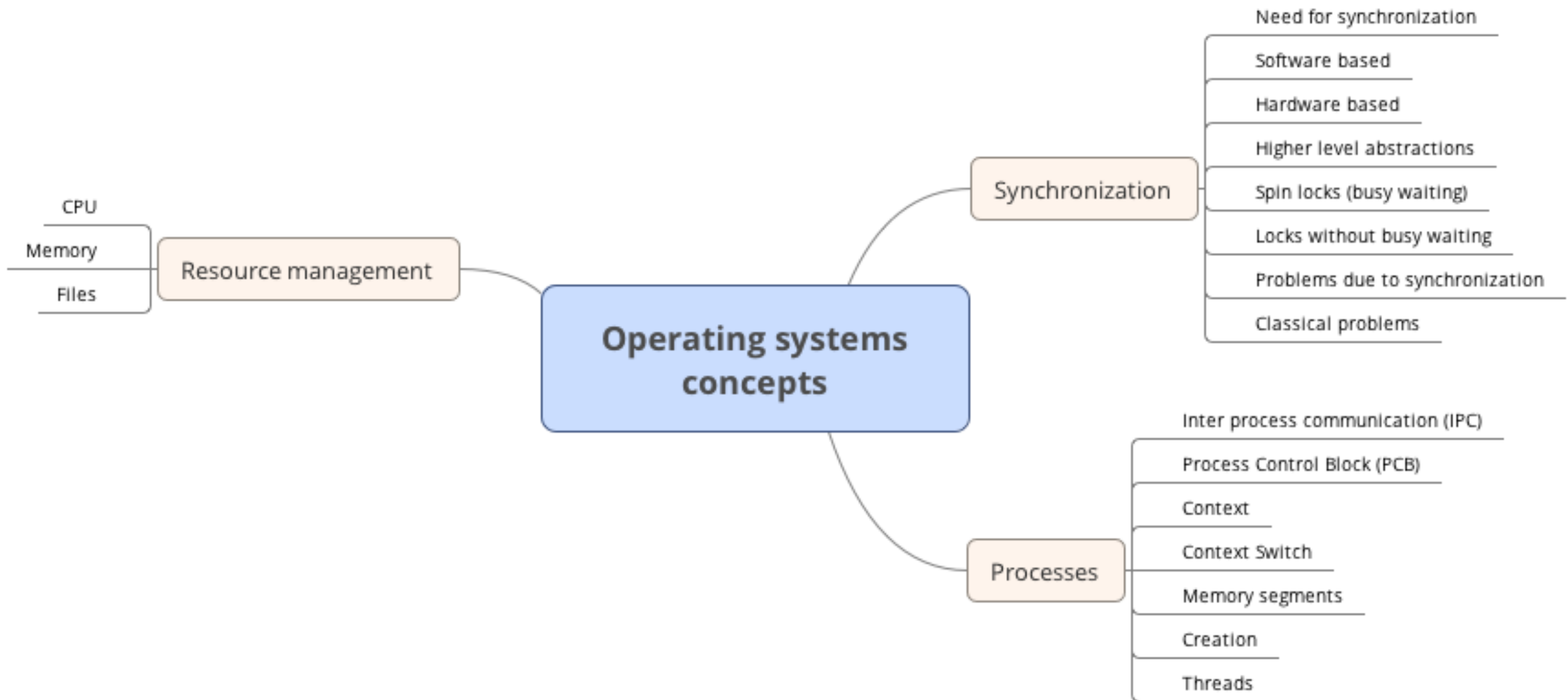
Operating system concepts (2)

Important concepts and how they relate to each other.



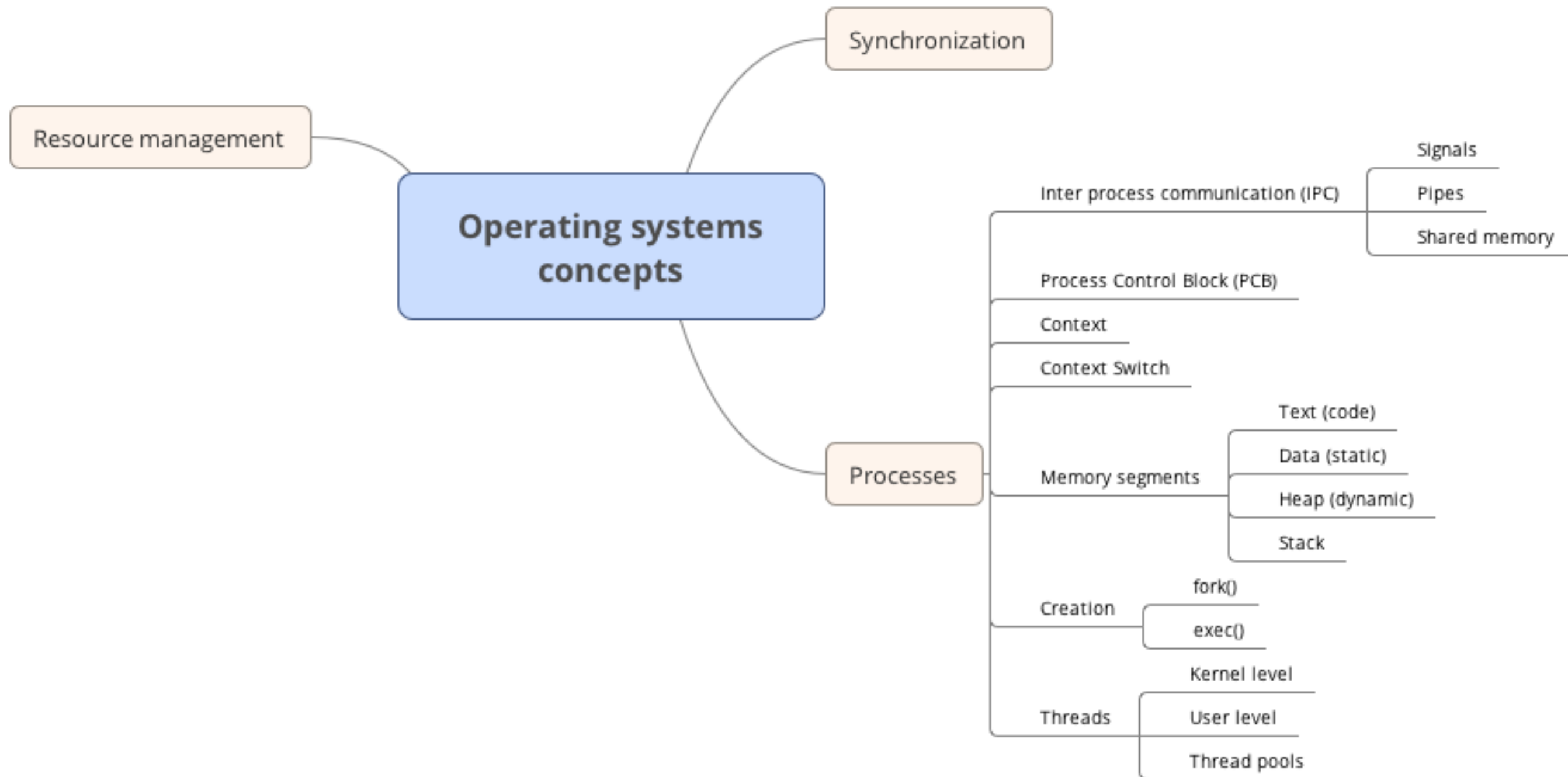
Operating system concepts (3)

Important concepts and how they relate to each other.



Operating system concepts (4)

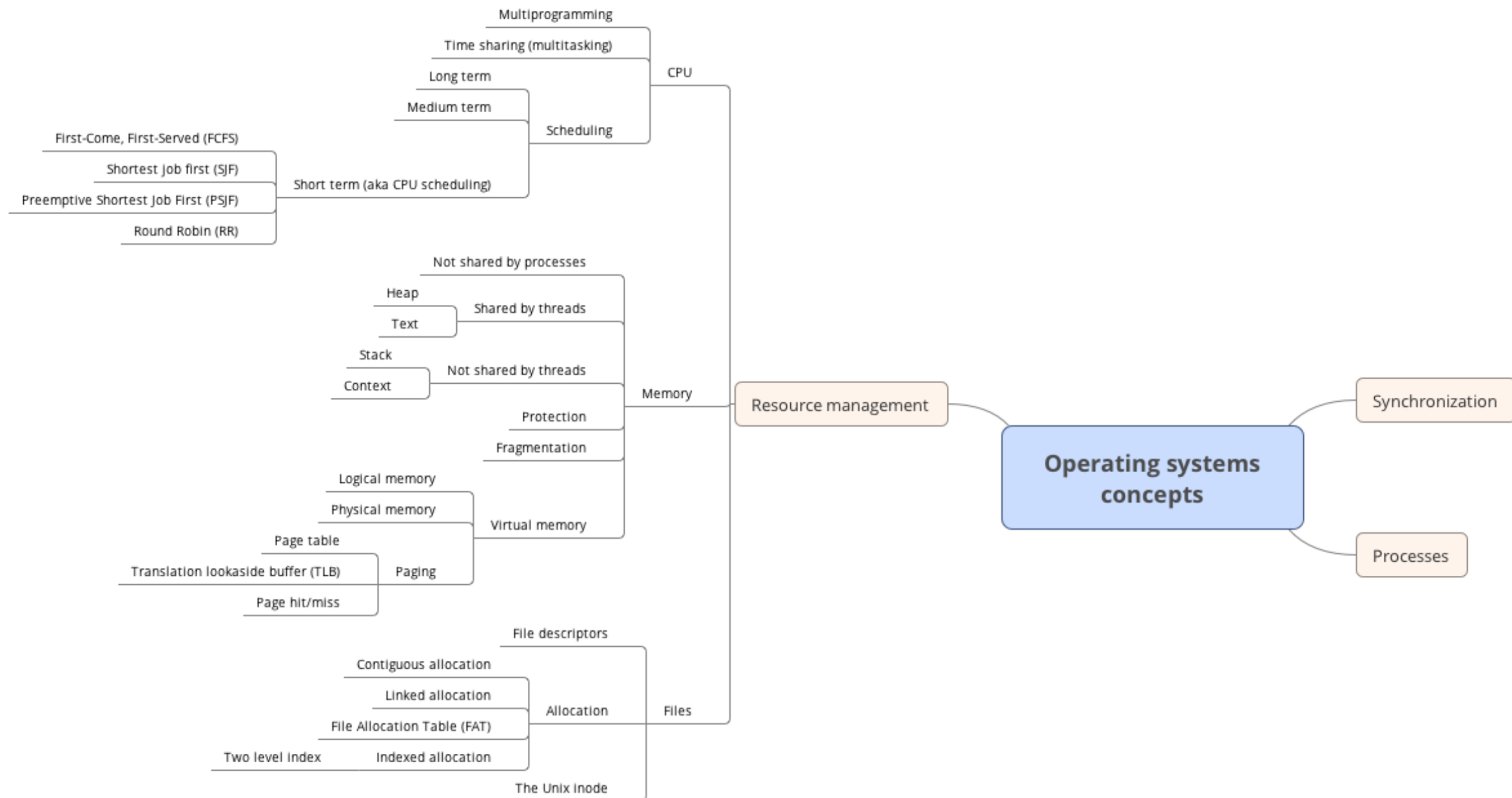
Important concepts and how they relate to each other.



Operating system concepts

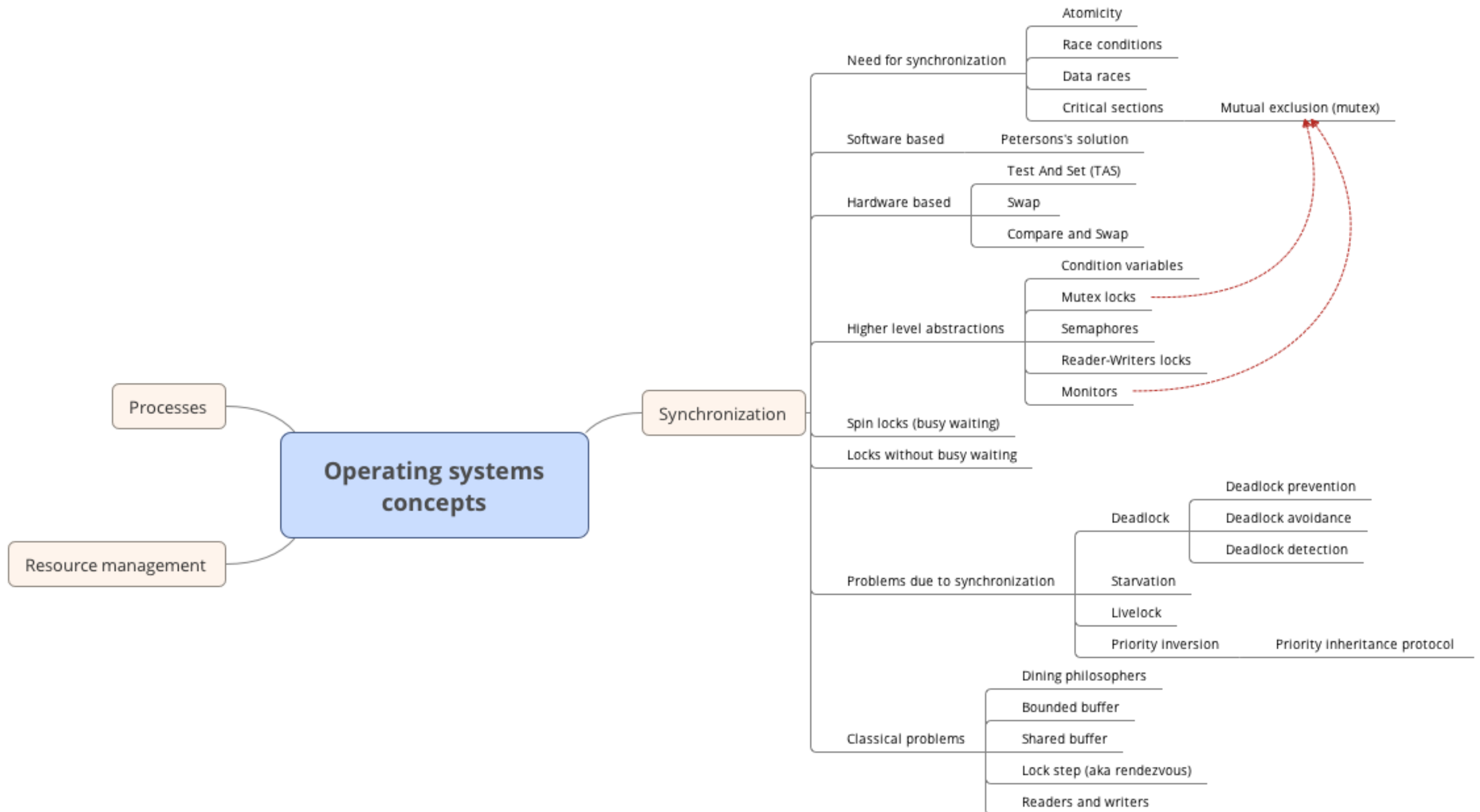
(5)

Important concepts and how they relate to each other.



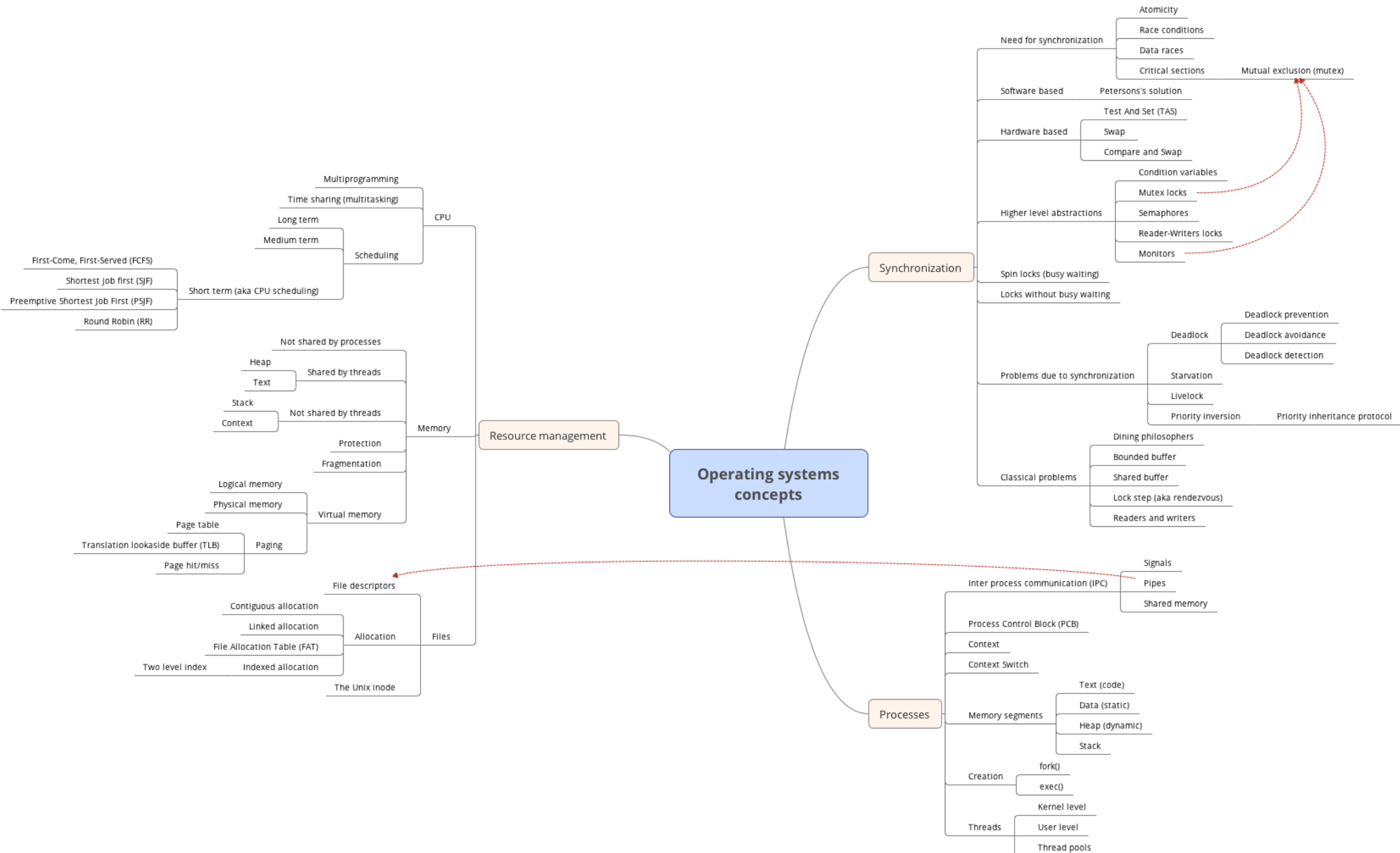
Operating system concepts (6)

Important concepts and how they relate to each other.



Operating system concepts (7)

Important concepts and how they relate to each other.



Concurrency models

Concurrency models

(1)

A number of formalisms for modeling and understanding concurrent systems have been developed.

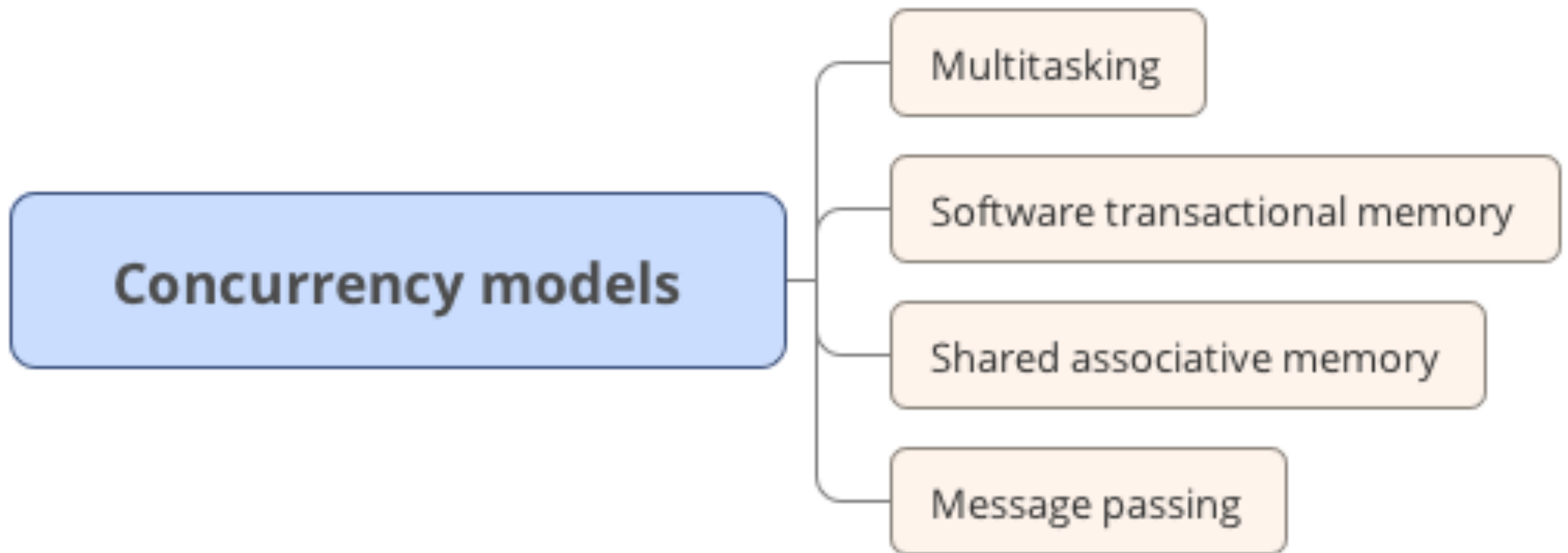


Concurrency models

Concurrency models

(2)

A number of formalisms for modeling and understanding concurrent systems have been developed.



Message passing

Message passing can be classified into
two categories:

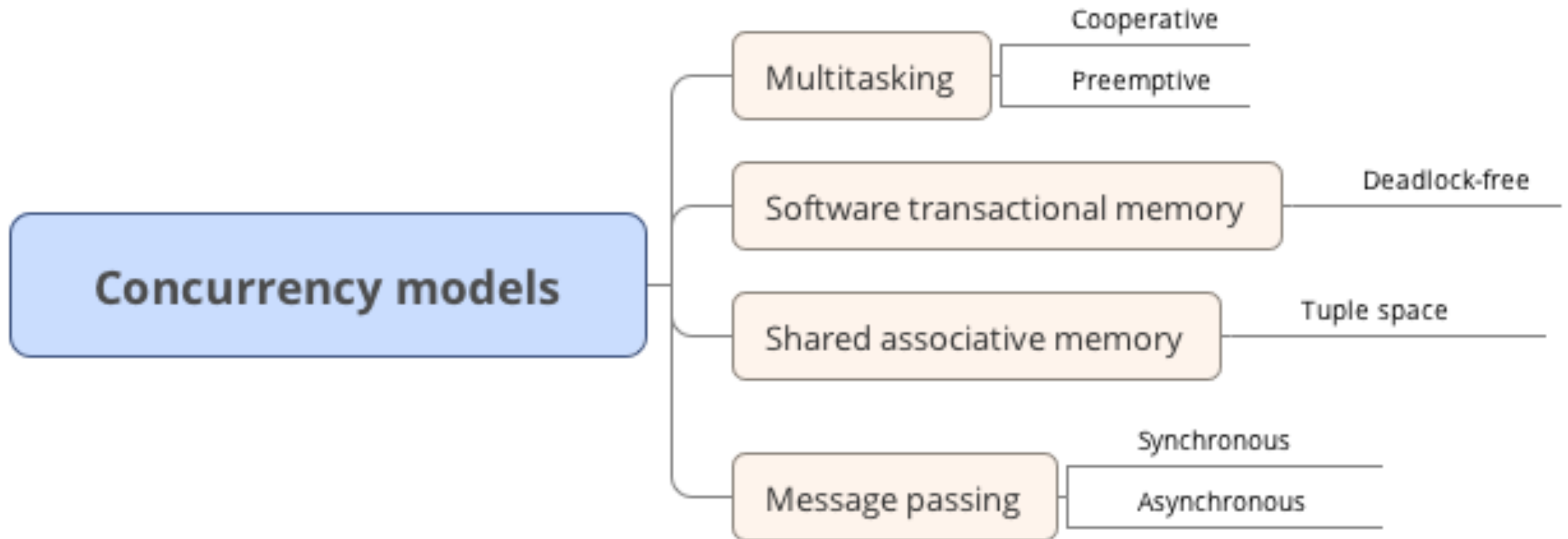
Synchronous

Asynchronous

Concurrency models

(3)

A number of formalisms for modeling and understanding concurrent systems have been developed.



Synchronous message passing

Your code sends a message, calls a function etc. and is blocked until an answer, a return value etc. arrives.

Calling someone on **telephone** is synchronous.

Asynchronous message passing

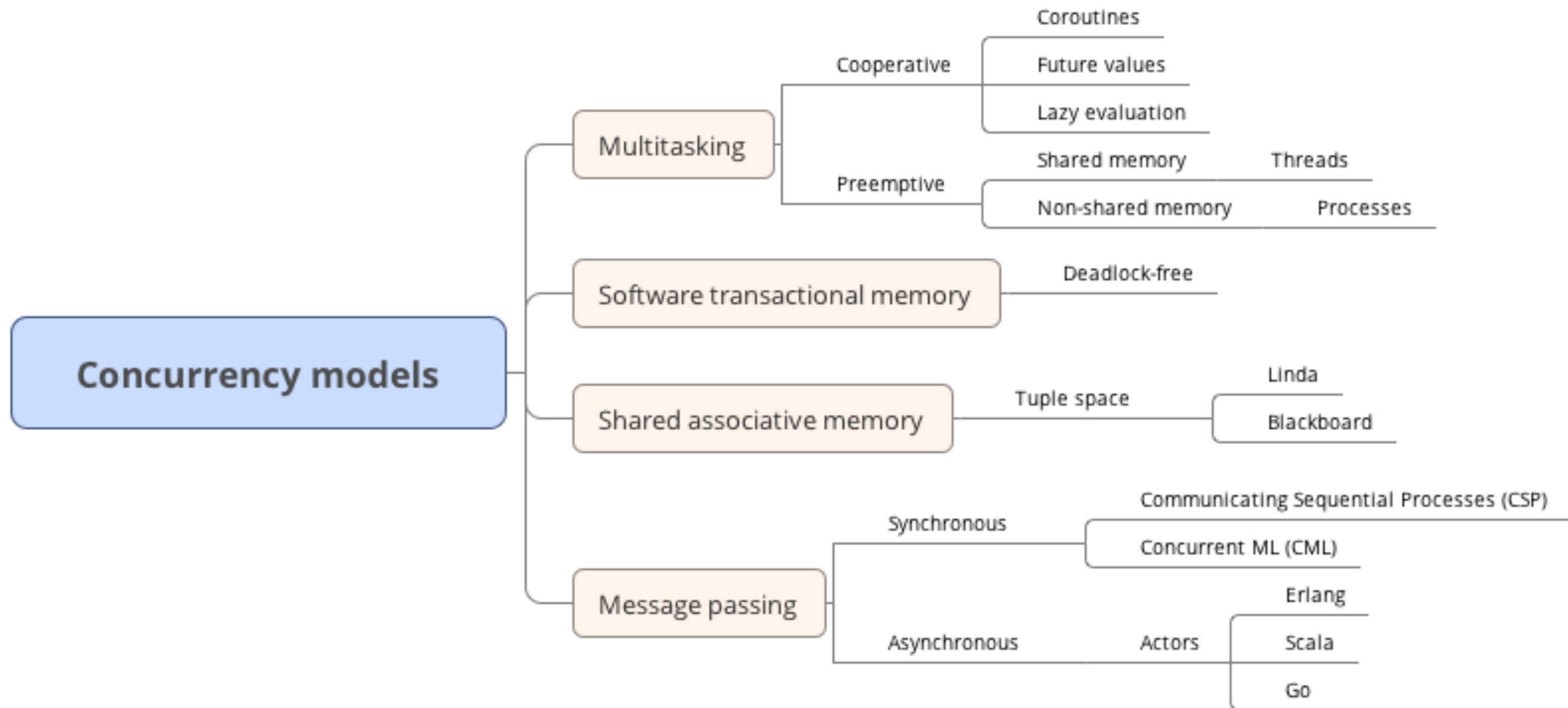
Your code continues executing after sending a message/calling a function, you usually pass a reference to a callback function that executes when the answer arrives (can happen in an hour, couple of days, years), your main thread continues in the meantime.

Communicating with someone per **mail** (or SMS, or online messaging) is asynchronous.

Concurrency models

(4)

A number of formalisms for modeling and understanding concurrent systems have been developed.

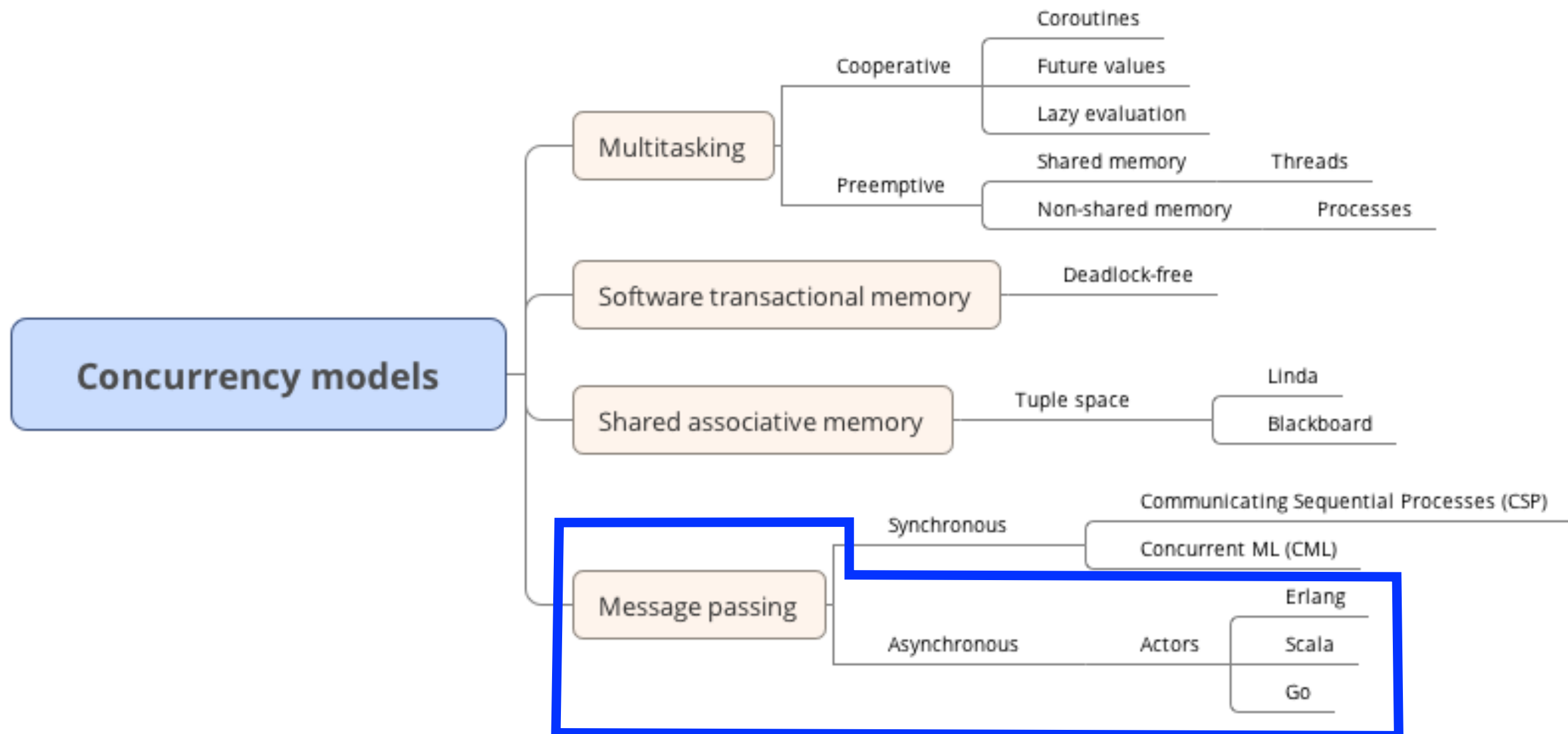


The actor model

The actor model

(1)

The actor model in computer science is a mathematical **model of concurrent computation** that treats **actors** as the universal primitives of concurrent computation:



The actor model

(2)

The Actor model originated in 1973.

The development of the actor model was motivated by the **prospect of highly parallel computing machines** consisting of dozens, hundreds or even thousands of independent microprocessors, each with its own local memory and communications processor, communicating via a high-performance communications network.

Since that time, the advent of massive concurrency through **multi-core** computer architectures has **revived interest** in the Actor model.



ABBA's released their first album Ring ring in 1973.

The actor model

(3)

An alternative to shared memory concurrency (threads) is **message passing** concurrency.

An actor is a computational entity that, in response to a message it receives, **concurrently** can:

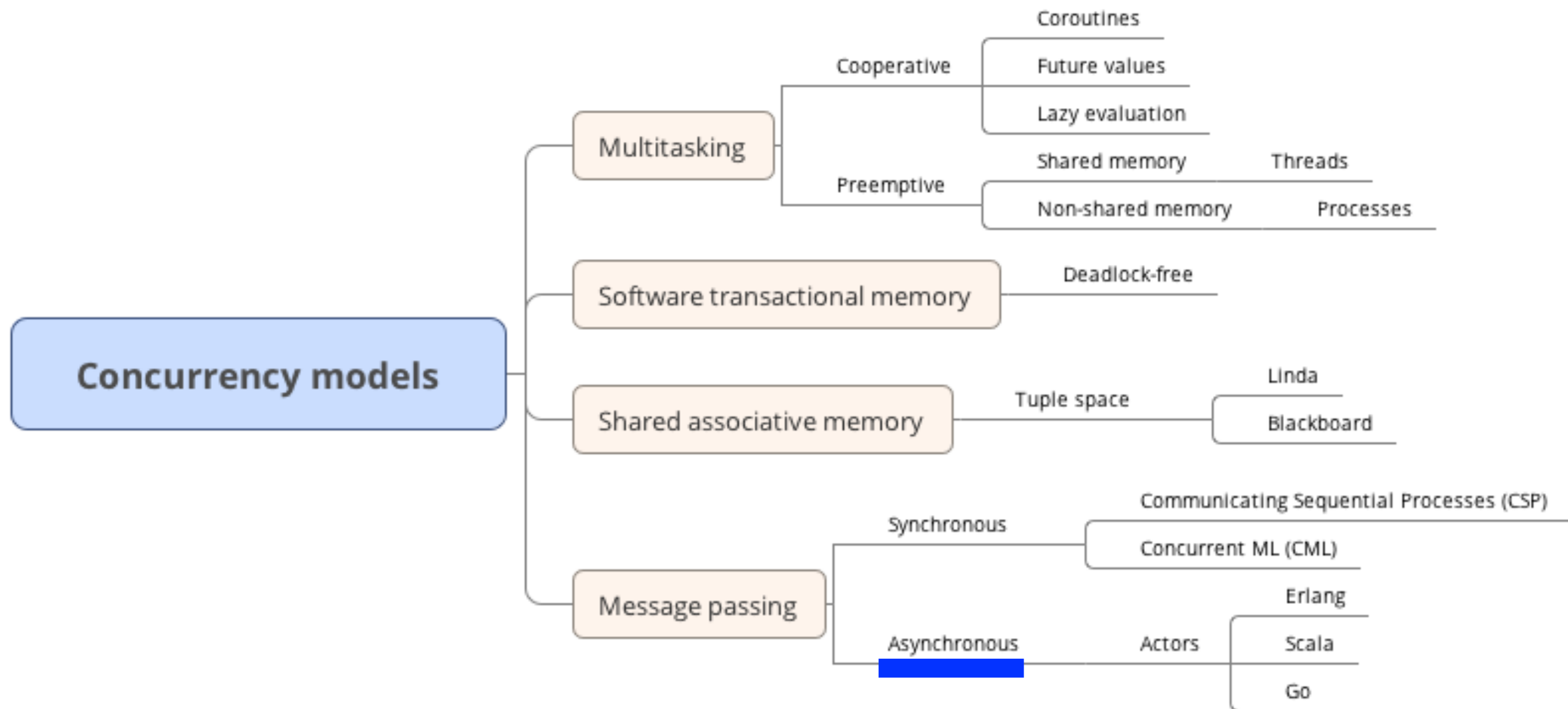
- * **send** a finite number of messages to other actors.
- * **create** a finite number of **new actors**.
- * **receive** messages from other actors.

There is no assumed sequence to the above actions and they could be carried concurrently.

The actor model

(4)

Decoupling the sender from communications sent was a fundamental advance of the Actor model enabling **asynchronous communication**.



The actor model

(5)

Recipients of **messages** are identified by **addresses**, sometimes called mailing addresses.

An actor can only communicate with actors whose addresses it has.

- * It can obtain those from a message it receives
- * or if the address is for an actor it has itself created.

The Actor model is characterized by:

- * inherent **concurrency of computation** within and among actors.
- * **dynamic creation of actors**.
- * **inclusion of actor addresses in messages**.
- * interaction only through direct **asynchronous message passing** with no restriction on message arrival order.

Actor creation plus the inclusion of the addresses of actors in messages means that Actors have a potentially **variable topology** in their relationship to one another.

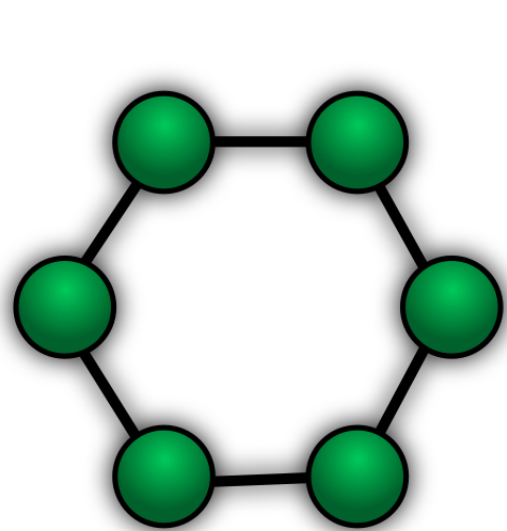
Topology

From Late Latin topologia, from Ancient Greek τόπος (tópos, “place, locality”) + -(o)logy (“study of, a branch of knowledge”).

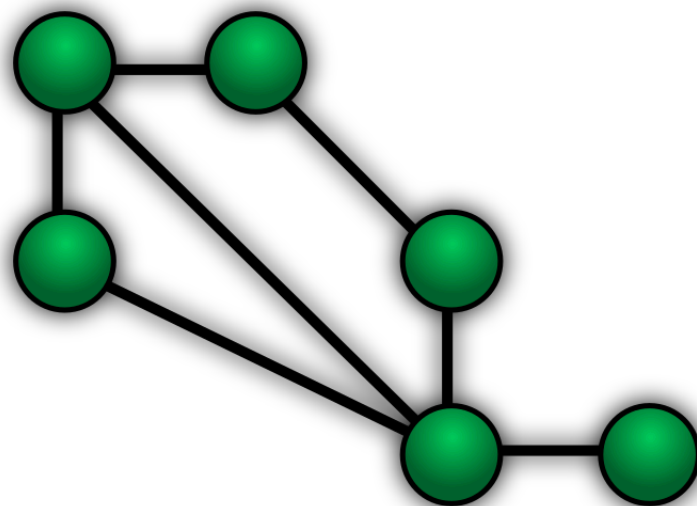
Topology

Context	Meaning
Mathematics	A branch of mathematics studying those properties of a geometric figure or solid that are not changed by stretching, bending and similar homeomorphisms.
Mathematics	A collection τ of subsets of a set X such that the empty set and X are both members of τ , and τ is closed under finitary intersections and arbitrary unions.
Medicine	The anatomical structure of part of the body.
Computing	The arrangement of nodes in a communications network.
Technology	The properties of a particular technological embodiment that are not affected by differences in the physical layout or form of its application.
Topography	The topographical study of geographic locations or given places in relation to their history.
Dated	The art of, or method for, assisting the memory by associating the thing or subject to be remembered with some place.

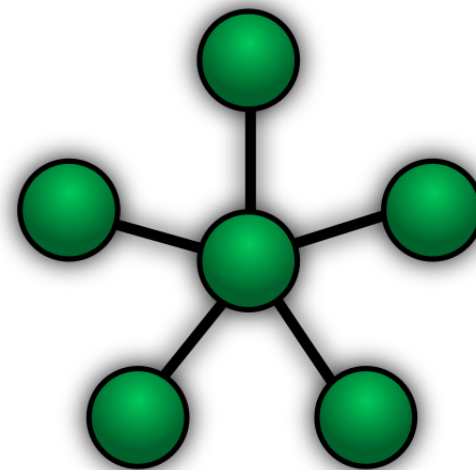
Examples of message passing topologies



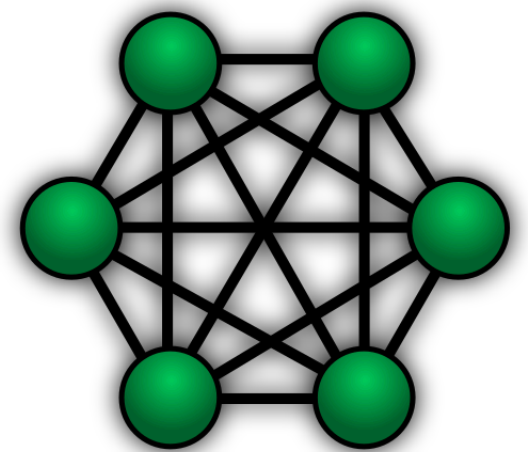
Ring



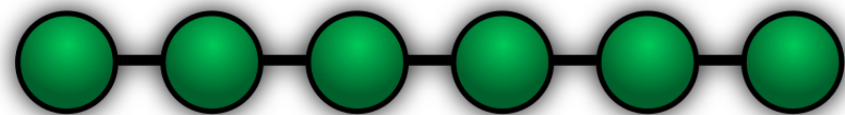
Mesh



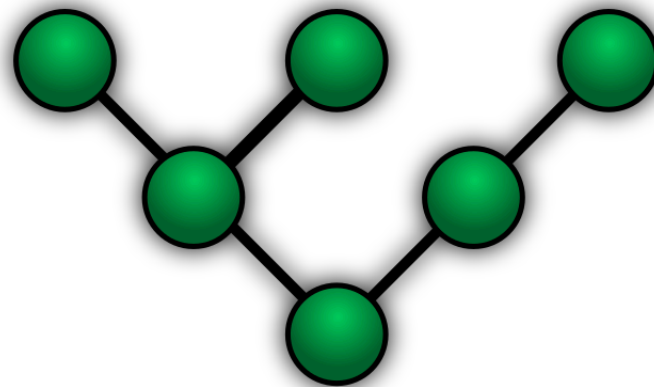
Star



Fully Connected



Line



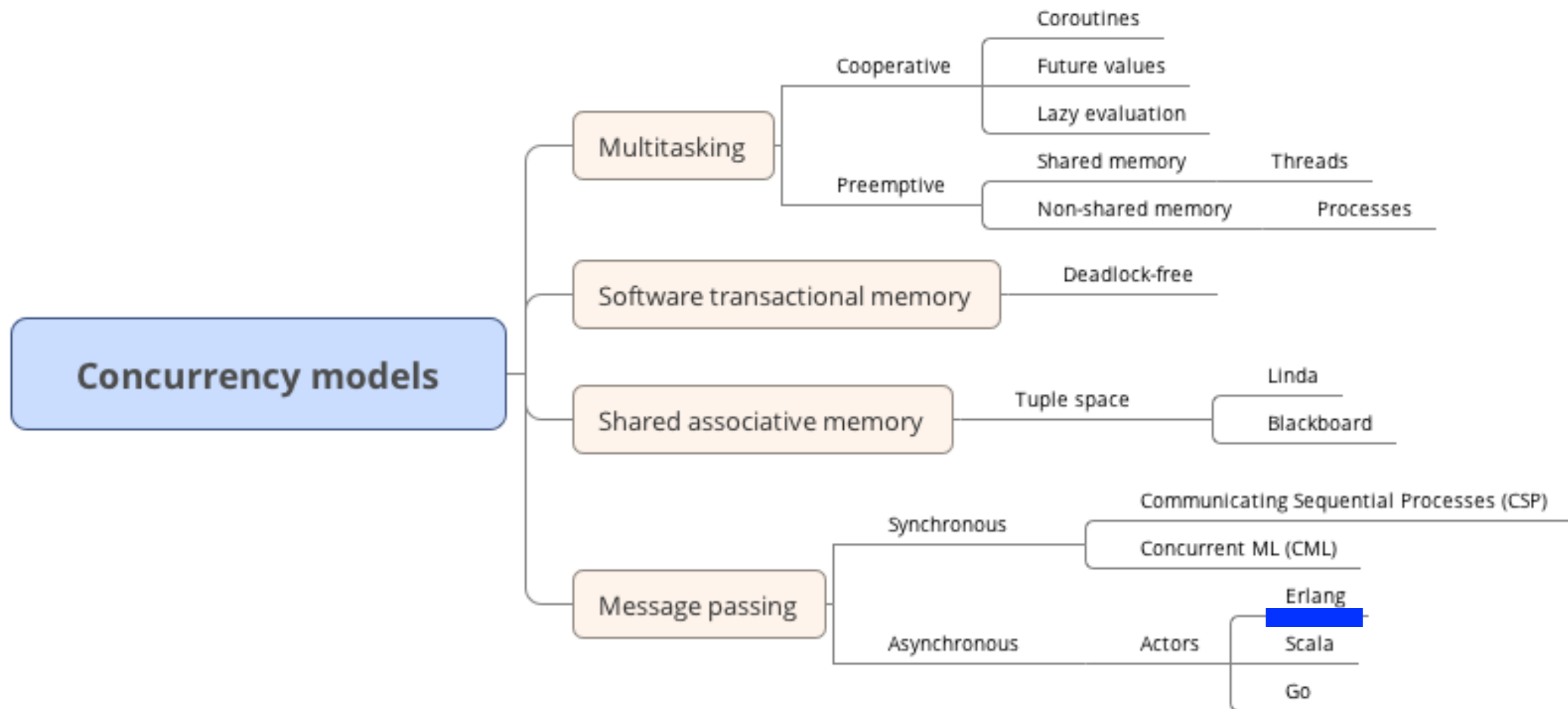
Tree

Erlang

Erlang is a programming language based on the actor model of concurrent computation.

Erlang

Erlang is a programming language based on the actor model of concurrent computation.



Erlang (programming language)

(1)

From Wikipedia, the free encyclopedia

Erlang is a general-purpose [concurrent](#), [garbage-collected programming language](#) and [runtime](#) system. The sequential subset of Erlang is a [functional language](#), with [strict evaluation](#), [single assignment](#), and [dynamic typing](#). For concurrency it follows the [Actor model](#). It was designed by [Ericsson](#) to support distributed, [fault-tolerant](#), [soft-real-time](#), non-stop applications. It supports [hot swapping](#), so that code can be changed without stopping a system.^[1]

Evaluation strategies

Eager evaluation

In computer programming, eager evaluation, also known as **strict** evaluation or **greedy** evaluation, is the evaluation strategy used by most traditional programming languages.

In eager evaluation, an expression is evaluated as soon as it is bound to a variable.

Imperative programming languages, where the order of execution is implicitly defined by the structure of the source code, almost always use eager evaluation.

Lazy evaluation

An opposite alternative to eager evaluation is lazy evaluation, where expressions are evaluated only when a dependent expression is evaluated depending upon a defined evaluation strategy.

Delayed evaluation is used particularly in functional programming languages.



An advanced, purely functional programming language

Haskell is one of the few languages that uses Lazy evaluation.

Erlang (programming language)

(2)

From Wikipedia, the free encyclopedia

While [threads](#) are considered to be a complicated and error-prone topic in most languages, Erlang provides language-level features for creating and managing processes with the aim of simplifying concurrent programming. Though all concurrency is explicit in Erlang, processes communicate using [message passing](#) instead of shared variables, which removes the need for [locks](#).

The first version was developed by Joe Armstrong in 1986.^[2] It was originally a proprietary language within Ericsson, but was released as [open source](#) in 1998.

The founding fathers of Erlang



Joe Armstrong is a co-inventor of Erlang. When at the Ericsson computer science lab in 1986, he was part of the team who **designed and implemented the first version of Erlang**. He has written several Erlang books including Programming Erlang Software for a Concurrent World. Joe held the first ever Erlang course and has taught Erlang to hundreds of programmers and held many lectures and keynotes describing the technology.

Joe has a PhD in computer science from the Royal Institute of Technology in Stockholm, Sweden and is an expert in the construction of fault tolerant systems. Joe was the chief software architect of the project which produced the Erlang OTP system. He has worked as an entrepreneur in one of the first Erlang startups (Bluetail) and has worked for 30 years in industry and research.



Robert Virding works for Erlang Solutions Ltd as a Principal Language Expert. While at Ericsson AB, Robert was one of the co-inventors of the Erlang programming language. As one of the original members of the Ericsson Computer Science Lab, he took part in the **original system design** and contributed much of the **original libraries**, as well as to the **current compiler**. While at the lab he also did a lot of work on the implementation of logic and functional languages and on garbage collection. He has also worked as an entrepreneur and was one of the co-founders of one of the first Erlang startups (Bluetail). Robert also worked a number of years at the Swedish Defence Materiel Administration (FMV) Modelling and Simulations Group. He co-authored the first book (Prentice-Hall) on Erlang, and is regularly invited to teach and present at conferences and universities worldwide.



Bjarne Däcker joined Ericsson in 1966 as programmer and systems analyst. In 1984 he set up the **Computer Science Lab** together with Mike Williams to explore, develop and **introduce new software technology in Ericsson** often in collaboration with university research. The CSLab pioneered things like **Unix, A.I., Lisp, Prolog** and workstations in Ericsson. Erlang was created at the CSLab by an initial team of Joe Armstrong, Mike Williams and Robert Virding. Bjarne organised the first Erlang User Conference in 1994. He has had various external commitments such as chairman of the steering committee of the Swedish national research programme in Computer Science 1987-1992 and member of the Evaluation Committee of European Union's ICT-Prize. The CSLab was closed in 2002 in the IT crash. Bjarne holds a Technology Licentiate degree of the Royal Institute of Technology in Stockholm and was promoted to an Honorary Doctorate of Technology at Linköping University in 1993. He is also a member of the Royal Swedish Academy of the Engineering Sciences.



Mike Williams is originally from South Wales, but has in fact lived in Sweden longer than he has anywhere else.

Way back in the 1960's after working as a Atheistic Missionary in Malawi, Mike went to Cambridge where he learnt a lot about drinking beer and rather less about "Mechanical Sciences". He then moved to Sweden in 1970 (guess why :-)) and joined Ericsson as a hardware designer. The price of beer in Sweden being horrendously expensive enabled Mike to concentrate more on other things, He joined with Bjarne Däcker to found the Ericsson Computer Science Laboratory 1980. One of the things they did in the Computer Science lab was to "invent" Erlang. Mike's role was to develop the first **Erlang virtual machine** (Joe developed the compiler and machine architecture). He worked out the primitives for **fault handling** and **dynamic code replacement**

In 1990 Mike glided into management by a complete accident, and found he rather liked it. Since then he has been in charge of both large and small units within Ericsson which develop software.

Inside Erlang, The Rare Programming Language Behind WhatsApp's Success

Facebook's \$19 billion acquisition is winning the messaging wars thanks to an unusual programming language.

How do you support **450 million** users with only **32** engineers?

For WhatsApp, acquired earlier this week by Facebook, the answer is **Erlang**, a programming language developed in the '80s that is finally having its moment in the spotlight.

Erlang was developed by Swedish telecom giant Ericsson over 25 years ago, and now it's finding a home at messaging apps like **WhatsApp** and **TigerText**.

Even Facebook was singing the language's praises when it used Erlang to launch **Facebook Chat** back in 2009 — the same year it turned down the job application of WhatsApp cofounder Brian Acton.

“The language is very **expressive**,” says Igor Clark, a creative technologist. “You can talk at a high level and do quite a lot with its few key concepts.”

In practical terms, that initially made Erlang very good at efficiently executing commands across processors within a single machine. Fast-forward to 2014, and it’s making **Erlang** very good at executing an even more bewildering volume of commands across the global networks of servers we have come to know as **"the cloud."**

For use cases in gaming, financial services, and anything that mimics the **behaviour of a real-time** auction, that speed and **reliability**, at **massive scale**, is absolutely essential.

Equally attractive to engineering teams, and highly unusual, is the way that Erlang allows for **bug fixes and updates without downtime.**

This Erlang property is a legacy of the telecom imperative: As DePue says, "When you're on the phone, you can't have someone hang up on you because they're upgrading the system."

WEDNESDAY, AUGUST 16, 2006

Why Erlang Is a Great Language for Concurrent Programming

ABOUT ME



YARIV

**[VIEW MY COMPLETE
PROFILE](#)**

ABOUT ME

**I work at Facebook but the opinions I
publish in this blog are my own.**

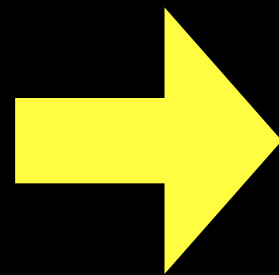
One of the greatest aspects of Erlang's concurrency is the way it's implemented behind the scenes.

Erlang code runs in a **virtual machine** called BEAM, which is responsible for **spawning**, **scheduling**, and **cleaning up** Erlang **processes**, as well as **passing messages** between them.

Erlang processes are much more **lightweight** than OS threads, which means that you can **potentially spawn millions of processes on a single box**. Try that with a Pthreads-based library and your machine would croak after the first 4000-10000 threads.

Erlang also maps nicely onto multi-core CPUs - why is this? - precisely because we use **a non-shared lots of parallel processes model of computation.**

- No shared memory
- No threads
- No locks



Ease of running
on a parallel
CPU.

Erlang in the wild

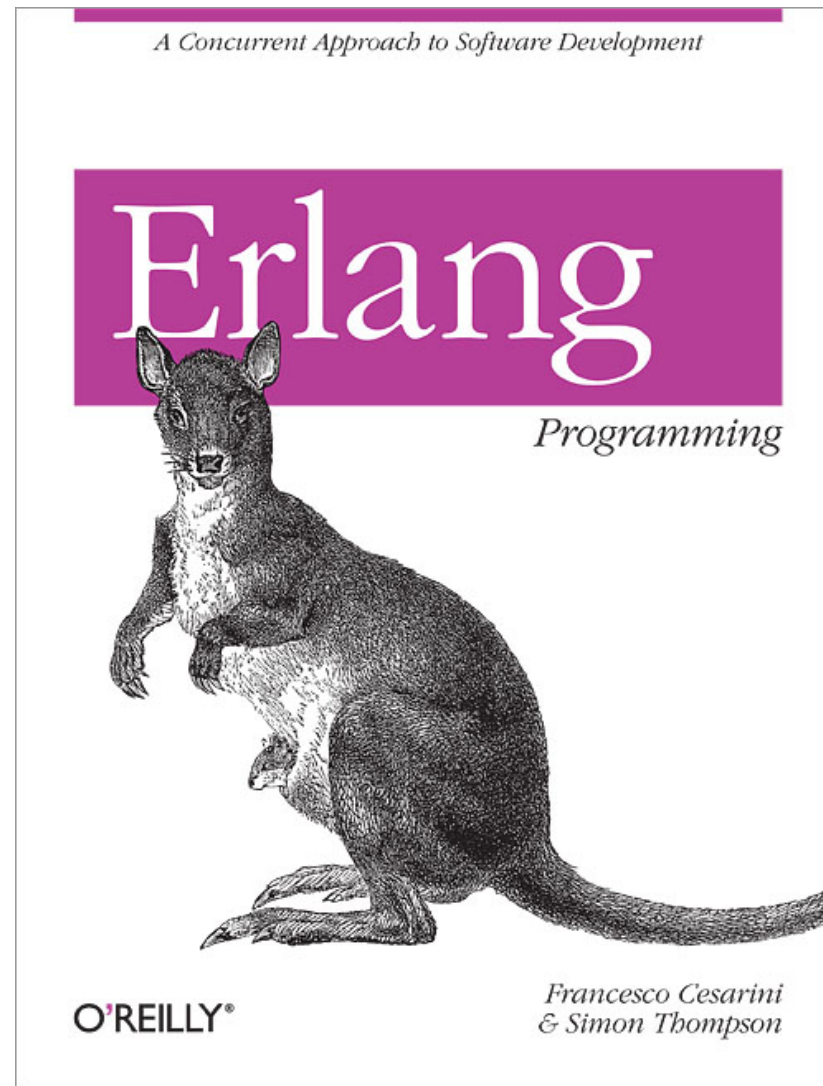
Examples of companies using Erlang in their production systems.

- **Amazon.com** uses Erlang to implement SimpleDB, providing database services as a part of the Amazon Web Services offering.
- **AOL's** digital advertising business is using Erlang for its real time bidding exchange systems.
- **Bet365**, the online gambling firm uses the language in production to drive its InPlay betting service, pushing live odds of sporting events to millions of customers in near real-time.
- **DNSimple**, a DNS provider that uses Erlang to run DNS servers in a globally distributed Anycast network, managing billions of requests per day.
- **Ericsson** uses Erlang in its support nodes, used in GPRS, 3G and LTE mobile networks worldwide.
- **Facebook** uses Erlang to power the backend of its chat service, handling more than 200 million active users. It can be observed in some of its HTTP response headers.
- **Goldman Sachs**, high-frequency trading programs
- **Huffington Post** uses Erlang for its commenting system on HuffPost Live.
- **Issuu**, an online digital publisher
- **Klarna**, a Swedish e-commerce company, has been using Erlang to handle 9 million customers and 50 million transaction since 2005.
- **Linden Lab** uses Erlang in its games.
- **Machine Zone**, a developer of Free-to-play games, uses Erlang in Game of War: Fire Age.
- **Motorola** is using Erlang in call processing products in the public-safety industry.
- **Smarmets**, sports betting exchange
- **T-Mobile** uses Erlang in its SMS and authentication systems.
- **Rackspace** uses Erlang in some of its internal applications to manage networking devices.
- **Rakuten** uses Erlang for its distributed file system.
- **Wargaming** uses Erlang for message delivery and communication between game players.
- **WhatsApp** uses Erlang to run messaging servers, achieving up to 2 million connected users per server.
- **Yahoo!** uses it in its social bookmarking service, Delicious, which has more than 5 million users and 150 million bookmarked URLs.

Erlang Programming

A Concurrent Approach to Software Development

Cesarin & Thompson

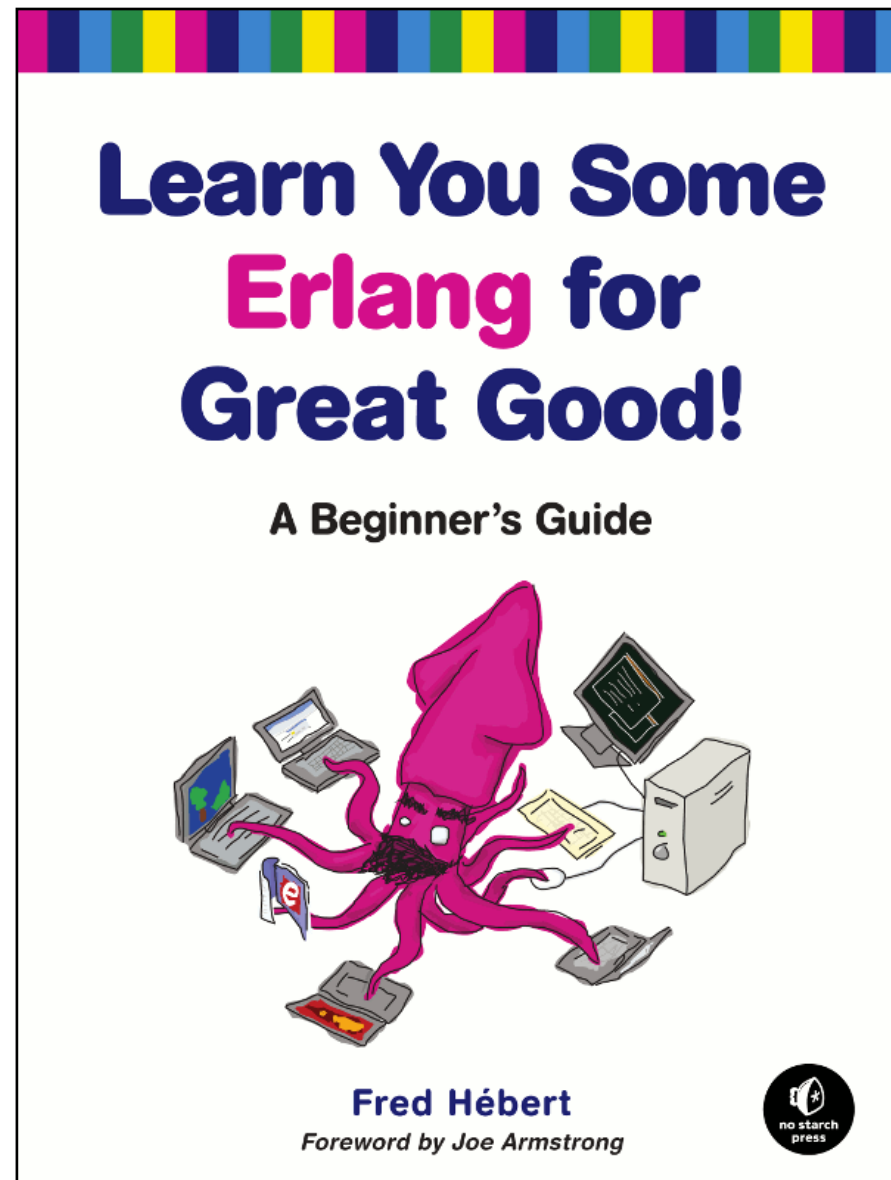


All you need to learn about Erlang is covered during the lectures and tutorials. This book is recommended for those who wish to learn more or prefer to have a text book as a complement to other resources.

Learn you some Erlang for Great Good!

A Beginner's Guide

Fred Hébert



Available for free online

<http://learnyousomeerlang.com>