Architecting System Performance; Greedy and Lazy Patterns

by *Gerrit Muller* TNO-ESI, University of South-Eastern Norway]
e-mail: gaudisite@gmail.com
www.gaudisite.nl

Abstract

Greedy and lazy are two opposite patterns in performance design. An extreme application of both patterns is start-up, where greedy starts as much as possible, and lazy as little as possible.

September 1, 2020
status: preliminary draft
version: 0.1

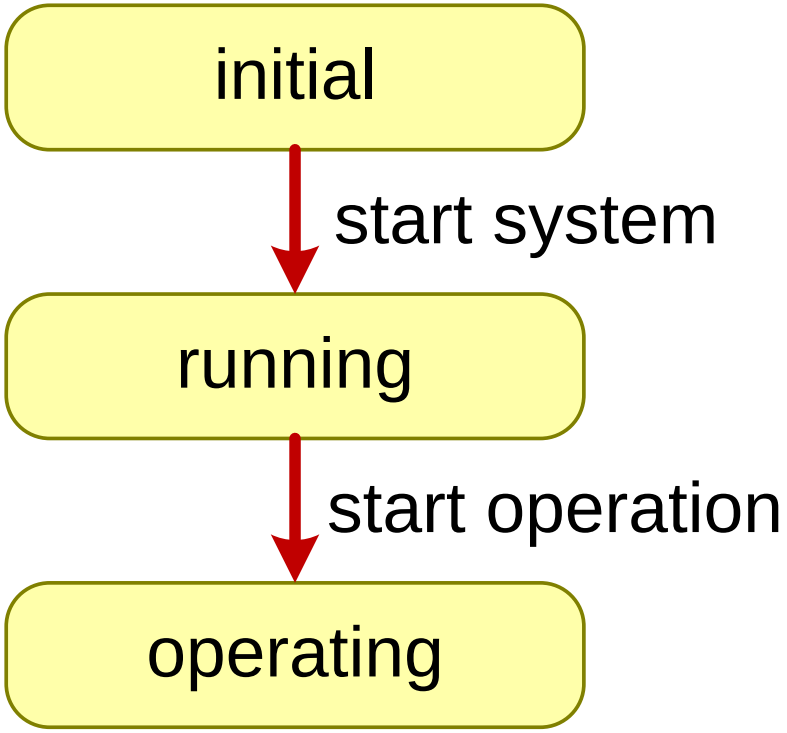
logo
TBD

Greedy and Lazy Patterns

	lazy (on demand, pull)	*greedy* (push, forecast)
what	do nothing until someone needs it	prepare time consuming operations, when resources are idle
benefits	no resource usage unless needed	results are available immediately
disadvantages	time to result depends on execution time	some resource use is wasted
when	default	to achieve required performance (explore other concepts too!)

this pattern applies to all domains (IT, goods flow, energy)

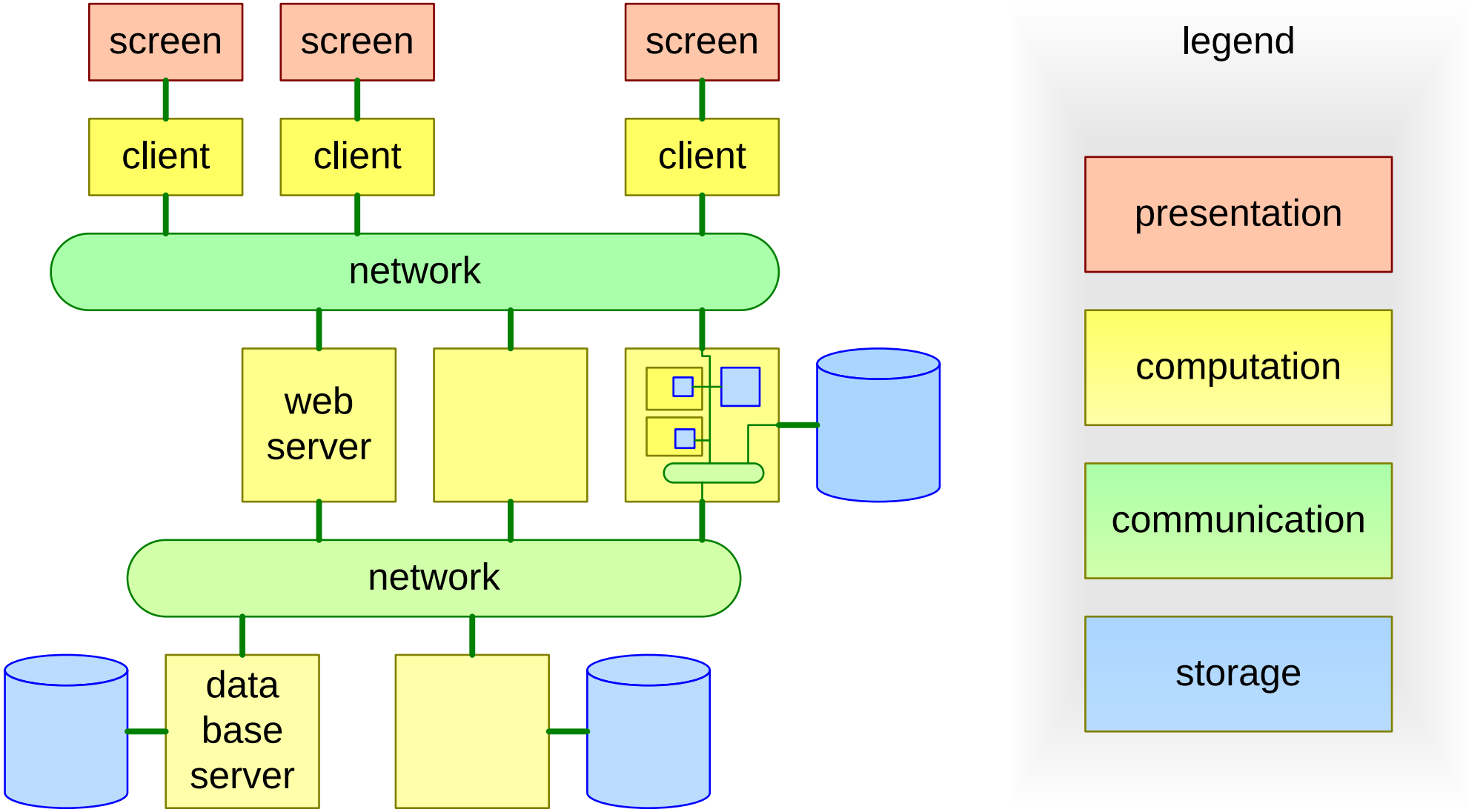
Start up of Systems as Example



How much time does it take
to start a laptop with Windows?

How much time does it take
to start an application (e.g. Word)?

Example from Cloud Applications



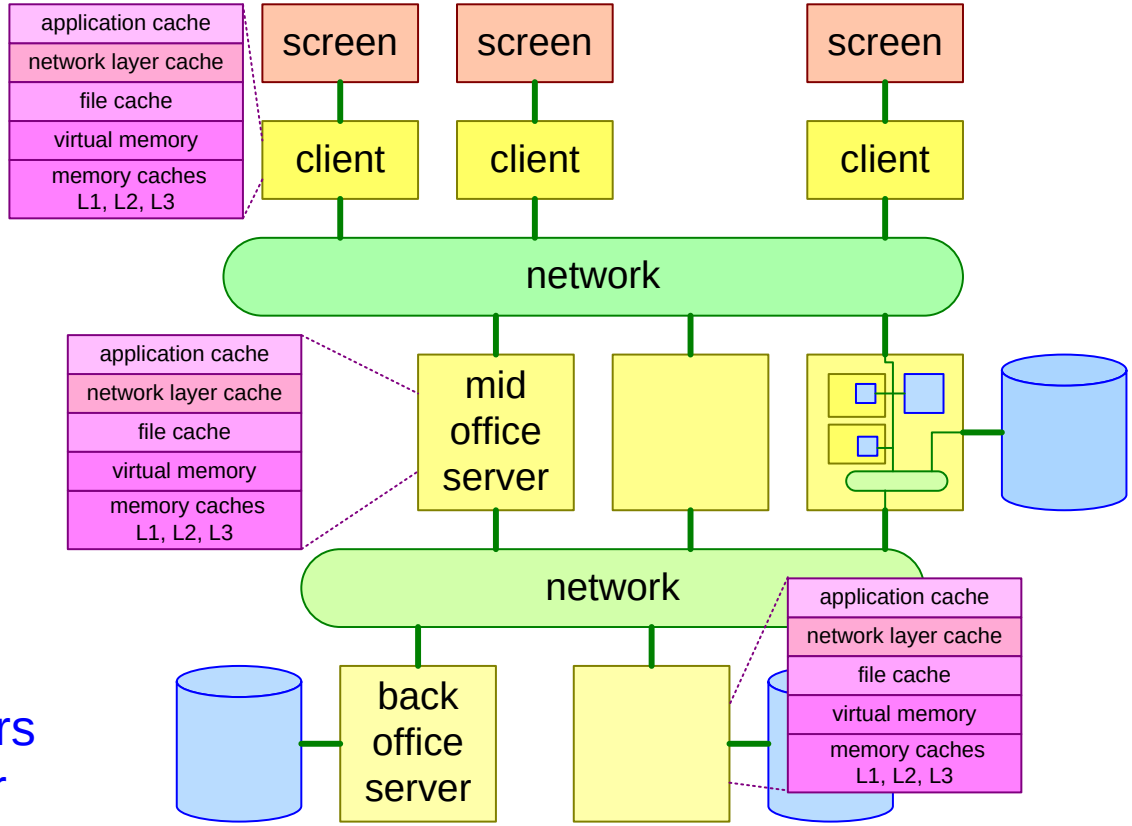
Caching Pattern (Physical Grab Stock)

performance issues	*solution patterns*	*design parameters*
	frequently used subset in fast local storage	caching algorithm
long latency (mass) storage	low latency	storage location
long latency communication	less communication	cache size
overhead communication	large chunks (less overhead)	chunk size
resource intensive processing	processing once (keep results)	format

Many Layers of Caching

	cache miss penalty	cache hit performance
application cache	1 s	10 ms
network layer cache	100 ms	1 ms
file cache	10 ms	10 µs
virtual memory	1 ms	100 ns
memory caches L1, L2, L3	100 ns	1 ns

typical cache 2 orders of magnitude faster



Disadvantages of Caching Pattern

robustness for application changes

ability to benefit from technology improvements

robustness for changing context (e.g. scalability)

robustness for concurrent applications

failure modes in exceptional user space

These patterns increase **complexity** and **coupling.**

Use only when necessary for performance.