

Byte-Sized

Computer Science for Data People

Part 2: System Design



Topics We'll Cover

①

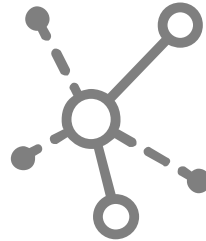
Clean Code



Writing performant code that others will be excited to reuse

②

System Design



Building systems and products that scale

③

Collaboration



Working productively with other people

Principles of System Design

User-Centricity



Designing your product around the people who will use it

Modularity



Splitting-up your teams and outputs to achieve better outcomes

Scalability



Making smart hardware and software choices to accommodate future needs

Deep-Dive: User-Centricity

Big Ideas

- **Be empathetic; understand through open-ended questions**
- **Design your product around user behaviors**
- **Don't give users more than they need**

Benefits

- More time to spend on things that people are actually asking for
- System is easier to adopt and maintain
- Fewer bugs in production caused by loose ends



Letting managers dictate the design of a product upfront that they'll never directly use



Designing around specific user experiences, iterating to incorporate their feedback

Related CS Concepts

- Five Why's
- Human-Centric Design
- Encapsulation & Law of Demeter

Deep-Dive: Modularity

Big Ideas

- **Every module / function should do only one thing**
- **Modules should interact using tightly-defined 'interfaces'**
- **Teams should be similarly organized, collaborating at their intersections**

Benefits

- Easier to build on and reuse existing code
- Easier testing and debugging
- Greater sense of accountability, ownership, and autonomy



Trying to find a bug in one long script that does 15+ things

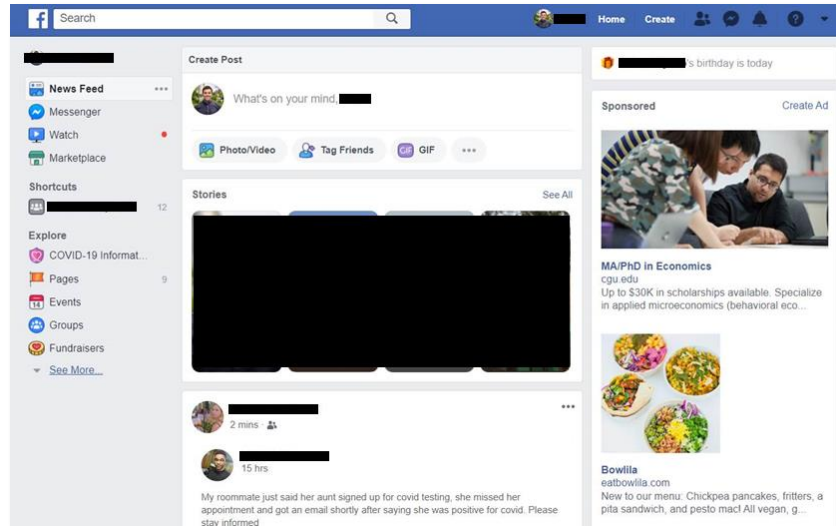


Writing fifteen smaller functions and unit tests so that you don't have to

Related CS Concepts

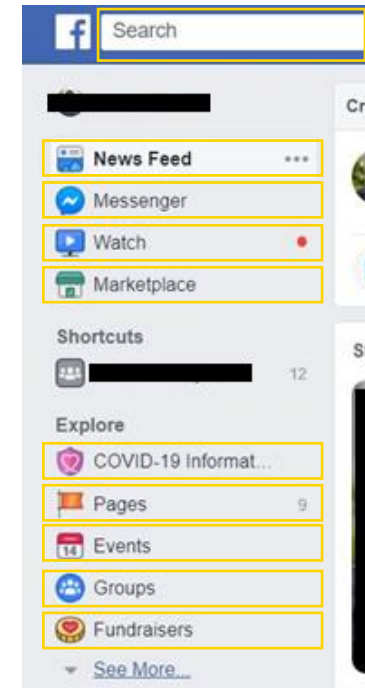
- Single-Responsibility Principle
- Design by Contract
- Orthogonality

Real-World Example



What users see

One cohesive product



What users don't see

Dozens of loosely-coupled, tightly-aligned teams owning individual features that share data through a shared hidden state

Sample Workflow

Import and call functions in main



Client1_main.py



Client2_main.py



...

Import

Write functions in .py scripts within /src



utilities.py



exploration.py



modeling.py



...

Push

Pull

Trigger automated tests with every push

main branch



test_main.py

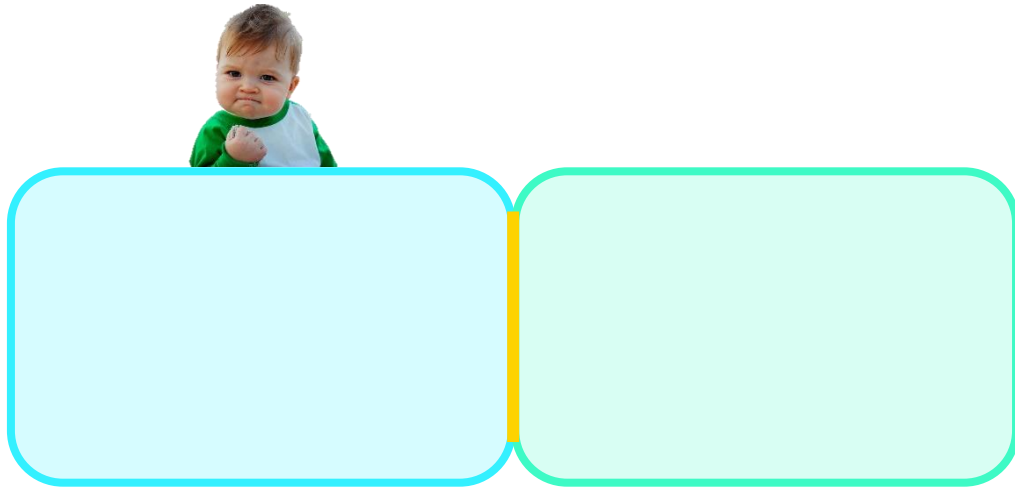
*Application
(calls functions)*



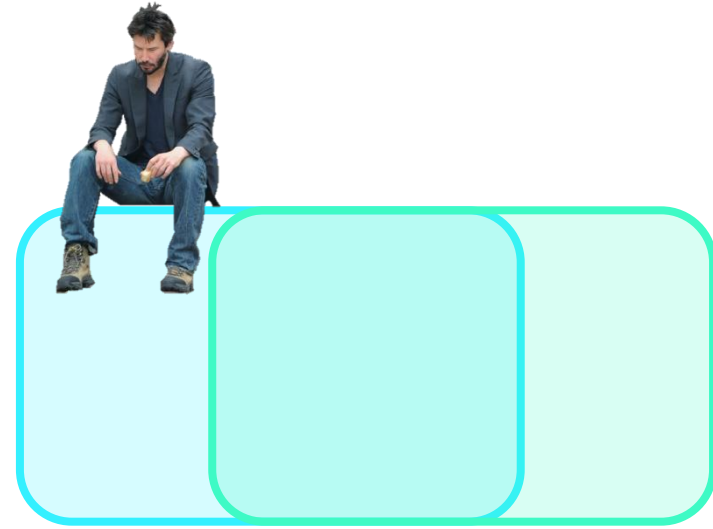
*Module
(defines functions)*



On “Integration”



Do define and test
interfaces between modules



Don't combine modules that are
designed around different behaviors

Deep-Dive: Scalability

Big Ideas

- **Ship product; don't sweat reversible design decisions**
- **Swap-out modules as your requirements change**
- **Profile cost, complexity, and space-time implications for big decisions**

Benefits

- Spend more time with users, less on technology
- Scale your product without rewriting code
- Make data-driven decisions regarding your architecture



Being tempted by fashionable technology (looking at you, k8s!)

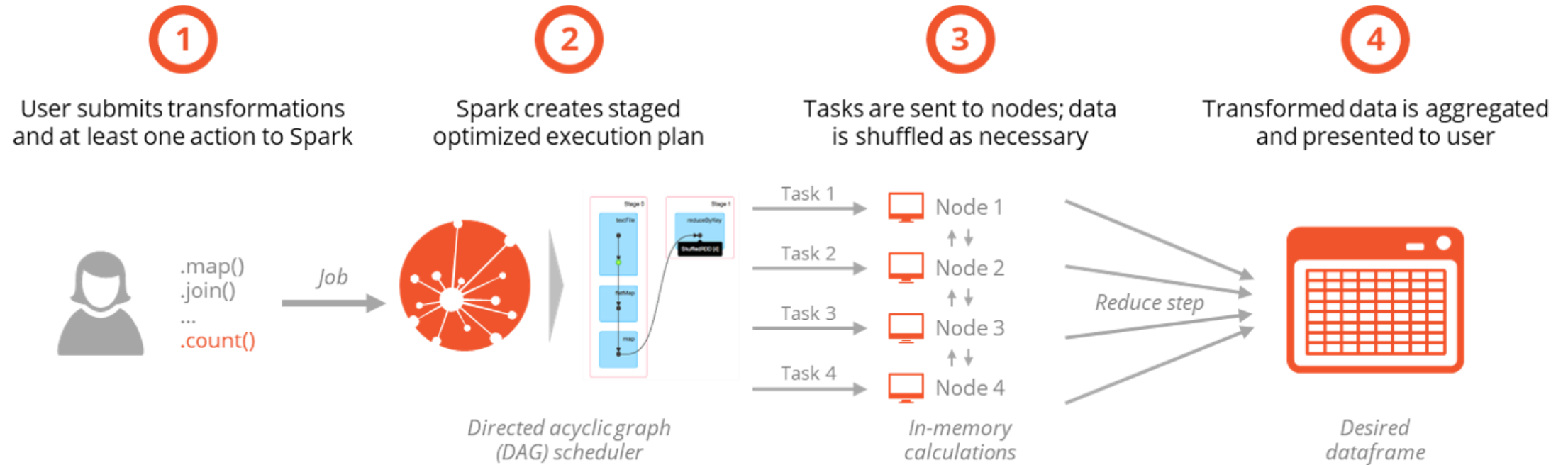


Keeping it simple until your requirements demand an upgrade

Related CS Concepts

- Shiny Object Syndrome
- Horizontal Scaling
- Bottlenecks (e.g., compute time, bandwidth, serialization, schedulers)

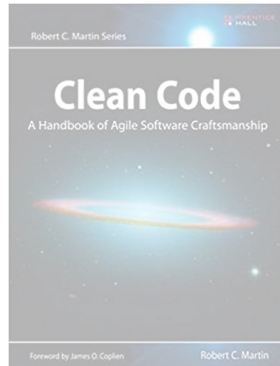
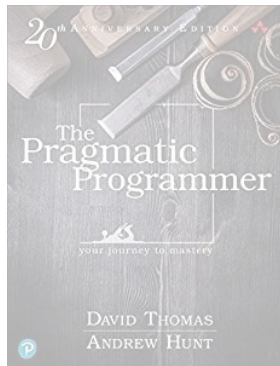
Horizontal Scaling Example: Spark



Book Recommendations

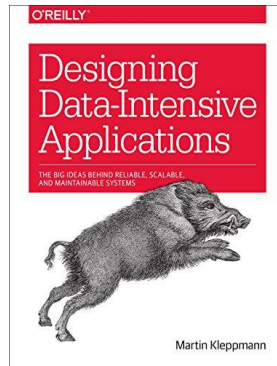
①

Clean Code



②

System Design



③

Collaboration

