[⬆] General Questions:

- What is *polymorphism*? (Variable of type Shape could refer to an object of type Square, Circle... Ability of a function to handle objects of many types)
- What is *encapsulation* (Packing of data and functions into a single component)
- What is *inversion of control*? (A design in which custom-written portions of a computer program receive the flow of control from a generic, reusable library)
- What is tail recursion? (A tail call is a subroutine call performed as the final action of a procedure)
- What is *virtual function*?
- What is *virtual method table*?
- What is *dynamic binding*?(Actual method implementation invoked is determined at run time based on the class of the object, not the type of the variable or expression)
- How does *garbage collector* work? (Mark and sweep: mark: traverse object graph starting from root objects, sweep: garbage collect unmarked objects. Optimizations: young/old generations, incremental mark and sweep)
- What is *semantic versioning*? (http://semver.org)
- What is *A/B testing*?

[⬆] Architecture:

- *Design principles*. (SOLID, DRY, KISS, YAGNI, Worse is better, convention over configuration, separation of concerns, principle of least knowledge, tourist principle, single source of truth, single version of the truth)
- Drawbacks of not using *separation of concerns*
    - Adding new features will take an order of magnitude longer
    - Impossible to optimize
    - Extremely difficult to test
    - Fixing and debugging can be a nightmare (fixing something in one place can lead to something else breaking that seems completely unrelated).

- *Microservices* are a style of software architecture that involves delivering systems as a set of very small, granular, independent collaborating services.
- Pros of *microservices* (The services are easy to replace, Services can be implemented using different programming languages, databases, hardware and software environment, depending on what fits best)
- What is *Docker*?
- *The Twelve-Factor App* (http://12factor.net)
- *The Reactive Manifesto* (http://www.reactivemanifesto.org)

Rule	Description
Single responsibility principle	A class should have one and only one task/responsibility. If class is performing more than one task, it leads to confusion.
Open/closed principle	The developers should focus more on extending the software entities rather than modifying them.
Liskov substitution principle	It should be possible to substitute the derived class with base class.

Interface segregation principle	It’s like Single Responsibility Principle but applicable to interfaces. Each interface should be responsible for a specific task. The developers should need to implement methods which he/she doesn’t need.
Dependency inversion principle	Depend upon Abstractions but not on concretions. This means that each module should be separated from other using an abstract layer which binds them together.

- *Design patterns*. (Creational:Builder,Object Pool,Factory Method,Signleton,Multiton,Prototype,Abstract Factory.Structural:Adapter,Bridge,Composite,Decorator,Facade,Flyweight,Proxy. Behavioral:Chain of Responsibility,Command,Interpreter,Iterator,Mediator,Memento,Observer,State,Strategy,Template Method,Visitor.)
- *Integration patterns*, SOA patterns.
- 3-tier architecture? (Presentation tier, Application tier, Data tier)
- 3-layer architecture? (DAO (Repository), Business (Service) layer, Controller)
- What is REST?
- What is *idempotent* operation? (The PUT and DELETE methods are referred to as idempotent, meaning that the operation will produce the same result no matter how many times it is repeated)
- What is *nullipotent* operation? (GET method is a safe method (or nullipotent), meaning that calling it produces no side-effects)
- Naked objects, Restful objects.
- What is *aspect-oriented programming*?

- Why do you need *application server*?
- *Inheritance* vs *Composition*.(Inheritance - is-a relationship, whether clients will want to use the subclass type as a superclass type. Composition - has-a or part-of relationship).
- *Multiple inheritance problem*.
- What is *uniform access principle*?(client code should not be affected by a decision to implement an attribute as a field or method)
- Advantages of using *modules*. (reuse, decoupling, namespace)

[⬆] WEB:

- WEB security vulnerabilities (XSS, CSRF, session fixation, SQL injection, man-in-the-middle, buffer overflow)
- CSRF prevention. (CSRF-token)
- What is *JSONP*, *CORS*? (A communication technique used in JavaScript programs running in web browsers to request data from a server in a different domain, something prohibited by typical web browsers because of the same-origin policy)
- HTTPS negotiation steps.
- What is HTTP Strict Transport Security (HSTS)? (Prevents Man in the Middle attacks)
- Browser-server communication methods: WebSocket, EventSource, Comet(Polling, Long-Polling, Streaming)
- What is *character encoding*?
- What is *role-based access controll* and *access control list*?
- What is session and persistent cookies, sessionStorage and localStorage?
- How to implement *remember-me*? (http://jaspan.com/improved_persistent_login_cookie_best_practice)

- Authentication using cookies, JWT (JSON Web Tokens).

[⬆] SQL:

- *SQL join types* (inner join, left/right outer join, full outer join, cross join, link)
- *SQL normal forms* (1.The domain of each attribute contains only atomic values, and the value of each attribute contains only a single value from that domain. 2.No non-prime attribute in the table is functionally dependent on a proper subset of any candidate key. 3.Every non-prime attribute is non-transitively dependent on every candidate key in the table. BCNF.Every non-trivial functional dependency in the table is a dependency on a superkey.)
- *Isolation levels* and Anomalies (Read Uncommitted, Read Committed, Repeatable Read, Serializable

Isolation_level\ Anomaly	**Lost_update (because of rollback)**	**Dirty_read**	**Non_repeatable_reads second_lost_update**	**Phantoms**	**Write_skew**
Read Uncommitted	-	may occur	may occur	may occur	may occur
Read Committed	-	-	may occur	may occur	may occur
Repeatable Read	-	-	-	may occur	may occur

Repeatable Read	-	-	-	may occur	may occur
Snapshot	-	-	-	-	may occur
Serializable	-	-	-	-	-

[⬆] NoSQL:

- Types of NoSQL databases?
  - Document Stores (MongoDB, Couchbase)
  - Key-Value Stores (Redis, Volgemort)
  - Column Stores (Cassandra)
  - Graph Stores (Neo4j, Giraph)

[⬆] Transactions:

- What ACID?
- What is 2-phase, 3-phase commit?
- What is pessimistic/optimistic locking?

[⬆] Scalability:

- Horizontal and vertical scaling.
- How to scale database? (Data partitioning, sharding(vertical/horizontal), replication(master-slave, master-master)).
- What is *synchronous multimaster replication*? (Each server can accept write requests, and modified data is transmitted from the original server to every other server before each transaction commits)

- What is *asynchronous multimaster replication*? (Each server works independently, and periodically communicates with the other servers to identify conflicting transactions. The conflicts can be resolved by users or conflict resolution rules)
- *Denormalization*.
- When to use messaging queue?
- Hadoop basics.
- storing sessions in Redis.
- MongoDB, Redis.

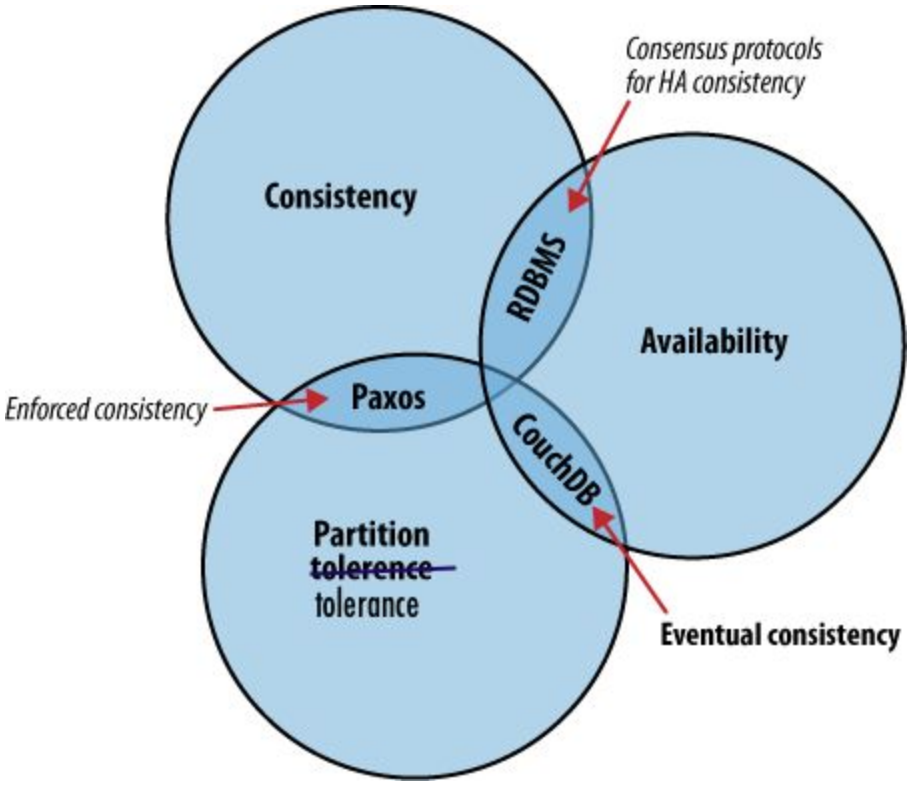
[⬆] Load balancing:

- What is *cloud computing*? (Cloud computing platform is a fully automated server platform that allows users to purchase, remotely create, dynamically scale, and administer system)
- sticky/non-sticky sessions

[⬆] Distributed:

- What is *CAP theorem*? (it is impossible for a distributed computer system to simultaneously provide all three of the following guarantees: *consistency*,

availability, *partition tolerance*)



- What is *map-reduce*? (Word count example)
- *Sharding counters*.
- Herlihy's consensus hierarchy. Every shared object can be assigned a consensus number, which is the maximum number of processes for which the object can solve wait-free consensus in an asynchronous system.

```
1 Read-write registers
2 Test-and-set, swap, fetch-and-add, queue, stack
⋮ ⋮
∞ Augmented queue, compare-and-swap, sticky byte
```

[⬆] Cache:

- What is *write-through* and *write-behind* caching? (write-through (synchronous), write-behind (asynchronous))

- HTTP cache options?

[⬆] Concurrency:

- What is *deadlock*, *livelock*? (Deadlock is a situation in which two or more competing actions are each waiting for the other to finish, and thus neither ever does. A livelock is similar to a deadlock, except that the states of the processes involved in the livelock constantly change with regard to one another, none progressing.)
- Deadlock avoidance. (prevention, detection, avoidance (Mutex hierarchy), and recovery)
- What is *starvation*? ()
- What is *race condition*? (Behavior of software system where the output is dependent on the sequence or timing of other uncontrollable events)
- What is *happens-before* relation?
- What is *thread contention*? (Contention is simply when two threads try to access either the same resource or related resources in such a way that at least one of the contending threads runs more slowly than it would if the other thread(s) were not running). Contention occurs when multiple threads try to acquire a lock at the same time
- What is a *thread-safe function*? (Can be safely invoked by multiple threads at the same time)
- Publish/Subscripe code
- What is *2-phase locking*? (Growing phase, shrinking phase. Guarantees serializablity for transactions, doesn't prevent deadlock).
- What is the difference between *thread* and *process*? (Threads (of the same process) run in a shared memory space, while processes run in separate memory spaces)

- What is *false sharing*, *cache pollution*, *cache miss*, *thread affinity*, *speculative execution*, *ABA-problem*?
- What is *lock-free* and *wait-free* algorithm?
- What is *sequential consistency*? (The result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program).
- What is *memory barrier*? (A memory barrier, also known as a membar, memory fence or fence instruction, is a type of barrier instruction that causes a CPU or compiler to enforce an ordering constraint on memory operations issued before and after the barrier instruction)
- Synchonization aids in Java
    - CountDownLatch
    - CyclicBarrier
    - Phaser
    - ReentrantLock
    - Exchanger
    - Semaphore
    - LinkedTransferQueue
- What is *data race*? (When a program contains two conflicting accesses that are not ordered by a happens-before relationship, it is said to contain a data race. Two accesses to (reads of or writes to) the same variable are said to be conflicting if at least one of the accesses is a write)
- Java *memory model*. (A program is correctly synchronized if and only if all sequentially consistent executions are free of data races. Correctly synchronized programs have sequentially consistent semantics. Causality requirement for incorrectly synchronized programs. https://dl.dropboxusercontent.com/u/1011627/journal.pdf)

- What is *monitor* in Java? (Each object in Java is associated with a monitor, which a thread can lock or unlock)
- What is *safe publication*?
- What is *wait*/*notify*?
- *Amdahl's law*? (Speedup = 1 / (1 - p + p / n))
- *Dining philosophers problem* (Resource hierarchy (first take lower-indexed fork), arbitrator, communication (dirty/clean forks)).
- *Produces/consumer* problem.
- *Readers/writers* problem.
- *Consensus number*. Maximum number of threads for which objects of the class can solve consensus problem.

[⬆] Networking:

- OSI model (Physical, Data link, Network, Transport, Session, Presentation, Application)
- Multithreading vs select
- Switch, hub, router.
- TCP congestion.

[⬆] Operating system:

- What is *memory mapped* file and its benefits?
- *Interprocess communication* methods. (Pipes, Events, Mailboxes/Ports (can be implemented by using shared memory and semaphores), Direct Message Passing).
- *Virtual memory* organization.

[⬆] Java:

- *WeakReference*, *SoftReference*, *PhantomReference*, *finalize()*, *ReferenceQueue*.
- How to correctly stop a thread? (Thread.interrupt())
- What is *Spring*? (Spring Framework is an application container for Java that supplies many useful features, such as Inversion of Control, Dependency Injection, abstract data access, transaction management, and more)
    - Spring is a framework for dependency injection: a design pattern that allows the developer to build very decoupled systems by injecting dependencies into classes.
    - It elegantly wraps Java libraries and makes then much easier to use in your application.
    - Included in the framework are implementations of commonly used patterns such as REST and MVC web framework which are predominately use by in web applications.
- *Garbage collection*. (Young/Old generation collectors combination examples: PS Scavenge/PS MarkSweep, Copy/MarkSweepCompact)
- What is *Hibernate* (Caches, lazy-loading)?
- How to write *benchmarks*?
- What is OSGI? (Specification describes a modular system and a service platform for the Java programming language that implements a complete and dynamic component model. Each bundle has its own classpath. Dependency hell avoidance. META-INF/MANIFEST.MF contains OSGI-info)
- What is JMS?
- Serializable / Externalizable
- What is a *servlet* (versions of servlet api)?
- What is a *servlet filter*? How to implement *GZipFilter*? (ResponseWrapper)
- What is *generics* and PECS (producer extends and consumer super)?
- What is *DAO* (Data Access Object)? (DAO allows one to switch between technologies like JDBC, Hibernate, JPA or JDO fairly easily)
- What is the difference between <?> and <Object>?

- XML: SAX, DOM, StAX

[⬆] Javascript:

- this keyword
- How *prototypes* work?
- inheritance
- differences between == and === (http://dorey.github.io/JavaScript-Equality-Table/)
- closures
- What is *MVC*, *MVP*, *MVVM*?
- What is *promise*?
- What is event *bubbling* and *capturing*? (target.addEventListener(type, listener[, useCapture]))
- What is *AMD*(Asynchronous Module Design) and *CommonJS*?
- What is *jQuery*?

[⬆] Codewriting:

- Implement binary search

```
int binarySearch(int[] a, int fromInclusive, int toExclusive, int key) {
    int low = fromInclusive;
    int high = toExclusive - 1;
    while (low <= high) {
        int mid = (low + high) >>> 1;
        int midVal = a[mid];
        if (midVal < key)
            low = mid + 1;
        else if (midVal > key)
            high = mid - 1;
        else
            return mid; // key found
```

```
    }
    return -(low + 1); // key not found
}
```

- Implement quick sort

```
void qSort(int[] a, int fromInclusive, int toInclusive) {
    int i = fromInclusive;
    int j = toInclusive;
    if (i >= j) return;
    int separator = a[i + random.nextInt(j - i + 1)];
    do {
        while (a[i] < separator) ++i;
        while (a[j] > separator) --j;
        if (i > j) break;
        int t = a[i];
        a[i] = a[j];
        a[j] = t;
        ++i;
        --j;
    } while (i <= j);
    qSort(a, fromInclusive, j);
    qSort(a, i, toInclusive);
}
```

- Implement permutations generation

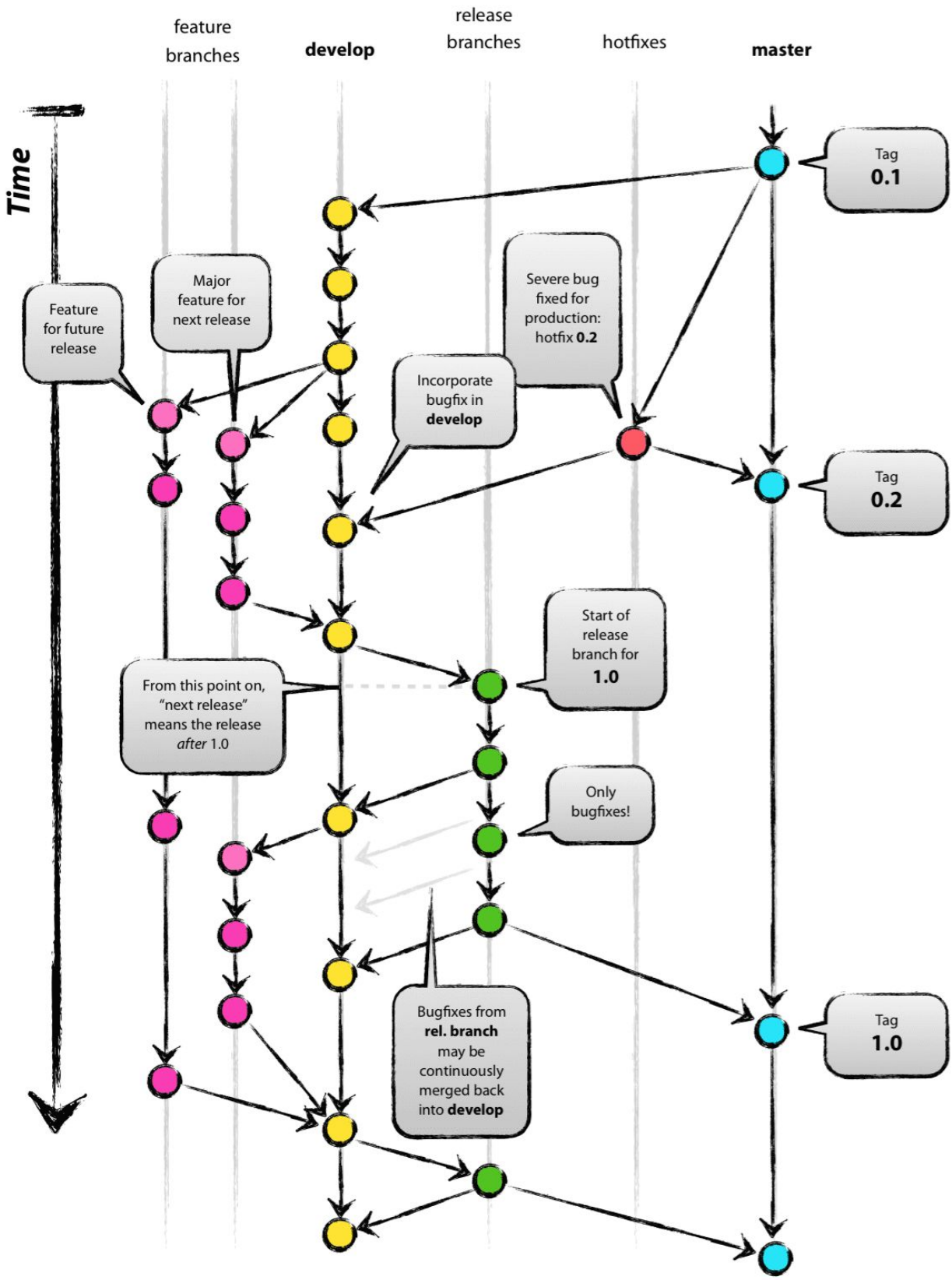
```
def generate_permutations(p, depth):
    n = len(p)
    if depth == n:
        yield p
    for i in range(n):
        if p[i] == 0:
            p[i] = depth
            yield from generate_permutations(p, depth + 1)
            p[i] = 0
```

```
for p in generate_permutations([0] * 3, 1):
    print(p)
```

[⬆] Git:

- *Git* workflow? (Master: production-ready state; Develop: latest delivered development changes for the next release; Feature Branches; Release

Branches; Hotfixes)


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

[⬆] DevOps:

- What is *Blue-green Deployment*, *Canary release*? (https://www.javacodegeeks.com/2016/02/blue-green-deployment.html)

[⬆] QA:

- What is *unit test*? (A test that purely tests a single unit of functionality)
- What is *integration test*? (Examine several parts of a system to make sure that when integrated, these parts behave as expected)
- Unit tests advantages?
- Types of tests: acceptance testing, functional testing, smoke testing, regression testing, unit testing, integration testing, stress testing, (Load, Performance, Sanity, Stability, Security, Feature, Progression, Installation, Business).
- Differences between stub and mock? (A stub is a test double with preprogrammed behavior. Mocks are stubs with preprogrammed expectations)
- Selenium tests and webdriver.
- How to test multithreading code?

[⬆] Agile:

- What is Agile? ()
    - Individuals and interactions over Processes and tools
    - Working software over Comprehensive documentation
    - Customer collaboration over Contract negotiation
    - Responding to change over Following a plan
- What is Scrum? (Roles: product owner, development team, scrum master. Events: sprint,
- What are the differences between Scrum and Waterfall? (http://www.leanagiletraining.com/agile/waterfall-versus-scrum-how-do-they-compare/)

- What is XP? ()
- What is Lean, Kanban?

[⬆] Algorithms:

- What O(n), Ω(n), Θ(n)?
- What is NP, NP-completeness, NP-hardness with examples?

[⬆] Other:

- How to find memory leak. (Memory snapshot diff).
- Profiling: sampling and instrumentation.
- Regular expressions. (Examples)
- XPath
- What are your goals to work in our company? (3 categories: professional, financial, social)
- What is *virtualization*?
- What is total/partial order?
- How to work with legacy code? (http://programmers.stackexchange.com/a/122024)

[⬆] Machine learning:

- Bayes' theorem. P(A|B) = P(B|A)P(A)/P(B), P(B) = sum(P(Ai)P(B|Ai))

[⬆] Cryptography:

- What is *public key cryptography*?
- What is *public key certificate*?
- *RSA*

```
select 2 primes: p,q
n = p*q
```

phi(n) = (p-1)*(q-1)

select 1<e<phi(n), gcd(e,phi(n))=1

d=e^-1 mod phi(n)

(e,n) - public key

(d,n) - private key

c = m^e mod n

m = c^d mod n = m^(e*d) mod n = m^(e*d mod phi(n)) mod n = m