Notus Vitae

When we are tired, we are attacked by ideas we conquered long ago.

Friedrich nietzsche

A musician wakes from a terrible nightmare. In his dream he finds himself in a society where music education has been made mandatory.

Paul Lockhart

Never trust a man a dog wouldn't

Internet

Live with a man 40 years. Share his house, his meals. Speak on every subject. Then tie him up, and hold him over the volcano's edge. And on that day, you will finally meet the man.

Firefly

Our deepest fear is not that we are inadequate. Our deepest fear is that we are powerful beyond measure.

Marianne Williamson

God, grant me the serenity to accept the things I cannot change,
Courage to change the things I can,
And wisdom to know the difference

Serenity Prayer

Anders Sildnes
andsild@gmail.com

Notes and Definitions

Abstract

An overview of algorithms, words and definitions from my BSc in computer science.

This document is **not** intended to be a *public friendly* document; it is written by one author and for one author, only. But you are still free to read.

I do not guarantee that the content is correct, errors may exist

Contents

List of Figures

List of Tables

Chapter 1

Words and Definitions

1.1 Algorithms

→ **Approximation algorithm.**
Find approximate solutions to Optimization problem

→ **Asymptotic Polynomial-time Approximation Scheme, APTAS.**
A family of algorithms, and a constant c such that all solutions has an approximation of $(1+\epsilon)\times OPT+c$ for minimization problems.

→ **Bin-packing.**
A series of algorithms to learn how to distribute n numbers into k bins. First-fit, best-fit, worst-fit (stack into where there is most free space), best-fit, etc.

→ **Complimentary slackness.**
Given an optimal solution to a linear program, Z_{LP} and it's dual Y_{LP}, with $x_1, x_2, \ldots x_n$ and $y_1, y_2, \ldots y_n$ respecively, with $w_1, w_2, \ldots w_n$ and $z_1, z_2, \ldots z_n$ as slack variables for each solution, respectively, then $\forall x, x_i z_i = 0$ and $\forall y, y_i w_i = 0$

This necessary condition for optimality conveys a fairly simple economic principle. In standard form (when maximizing), if there is slack in a constrained primal resource (i.e., there are "leftovers"), then additional quantities of that resource must have no value.

→ **Difference heuristic and approximation.**
While an heurisitc makes a choice, without a guarantee of optimality, an approximation can make a choice and know that this choice will render a solution within a factor of OPT.

→ **EDD.**
Earliest Due Date

→ **F-approximation.**
Also referred to as a linear approximation, using a function f, which is affine.

→ **FPTAS, fully polynomial approximation scheme.**
As in PTAS, Polynomial-time approximation scheme, just $\frac{1}{\epsilon}$. The algoritm is required to be polynomial both in running time and problem size.

Note that strongly NP-complete problems do not have any FPTAS.

→ **Integrality gap.**
The biggest difference between an IP and LP

→ **Locality of Reference.**
also known as the principle of locality, is a phenomenon describing the same value, or related storage locations, being frequently accessed. There are two basic types of reference locality âĂŞ temporal and spatial locality. Temporal locality refers to the reuse of specific data, and/or resources, within a relatively small time duration. Spatial locality refers to the use of data elements within relatively close storage locations. Sequential locality, a special case of spatial locality, occurs when data elements are arranged and accessed linearly, such as, traversing the elements in a one-dimensional array

→ **Makespan.**
The total length of a schedule; from 0 to $C_{\max}$

→ $\tilde{O}$.
Given function $f(x)$, $\tilde{O}(f(x)) = O(f(x) \cdot \log^k f(x))$

→ **Optimization problem.**
To find the best solution of n feasible solutions.

→ **Perfect matching.**
A collection $E' \subseteq E$ of edges in a graph $G = (V, E)$, such that $\forall v \in V$, are connected from E' only once.

→ **Pre-empty schedule.**
You can interrupt task and re-continue them.

→ **PTAS, Polynomial-time approximation scheme.**
Given an optimization problem (e.g. an NP-problem) and a parameter ϵ, produce a solution within $((1+\epsilon) \times OPT)$ $\epsilon > 0$
E.g. for the traveling salesman, a tour would be of length max $(1+\epsilon) \times L$, with L being the length of the tour
Note that for minimization, there is $1+\epsilon$, and for maximization, there is $1-\epsilon$
If you have a scheme with $(1 \pm \epsilon) \times OPT + \kappa$, then it is not under PTAS. PTAS only handles the former part, $(1 \pm \epsilon)$
For MAX SNP, there does not exist polynomial approximation schemes

→ **Parallel Random Access Machine.**
an abstract computer for designing parallel algorithms

→ **ρ-approximation.**
Polynomial algorithm that is guaranteed to have objective function to OPT within ρ of optimum (not the $(1+\epsilon)$ of PTAS, Polynomial-time approximation scheme).

→ **Scheduling.**
See Longest processing time rule,

- P_i = time to do a job i
- R_i = earliest time a job i can start
- C_i = time of completion for job i
- D_i = due date for job i
- $L_i = C_i - D_i$

→ **Strong duality.**
The optimal value of the dual is equal to that of the primal linear program.
$\sum y_i^* = \sum w_i x_i$

→ **Weak duality property.**
No dual program has a solution greater than the optimal of the primal linear program

→ **α approximation.**
Produce a solution who's value is within a factor of α of the optimal.

1.2 Calculus

→ **Adiabatic.**
adiabatic wall between two thermodynamic systems does not allow heat or matter to pass across it.

→ **A priori estimate.**
An estimate for a size of a solution or it's deriviates of a PDE. The estimate is made before the solution is known to exists.

→ **Affine.**
$f(x_1, x_2, \ldots, x_n) = a_1 x_1, a_2 x_2, \ldots a_n x_n$

→ **Analytic function.**
A function given by Convergence Power series.

Any polynomial, exponential and trigonometric function is analytic. Functions that are not differentiable at given points are not analytic.

→ **arcsin.**

$$\sin y = x$$
$$\arcsin x = \sin^{-1} x = y$$

Properties:

- $\arcsin x = \frac{\pi}{2} - \arccos x = 90 - \arccos x$
- $\cos(\arcsin x) = \sin(\arccos x) = \sqrt{1 - x^2}$
- $x = -1 \rightarrow \arcsin x = -1 \times \frac{\pi}{2}$
- $x = 1 \rightarrow \arcsin x = \frac{\pi}{2}$
- $x = 0 \rightarrow \arcsin x = 0$

→ **arccos.**
Properties:

- $x = -1 \rightarrow \arccos x = \pi$
- $x = 1 \rightarrow \arccos x = 0$
- $x = 0 \rightarrow \arccos x = \frac{\pi}{2}$

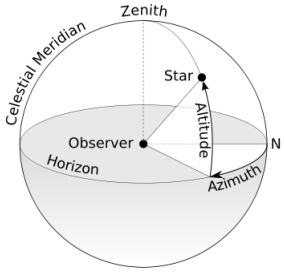
→ **Arc length.**
Length of a curve when straightened out.

→ **arccos.**
While cosine shows the relation between lengths in a triangle, arccos gives the angle.

→ **Arithmetic-geometric mean inequality.**

$$(\prod_{i=1}^{k} a_i)^{1/k} \leq \frac{1}{k} \sum_{i=1}^{k} a_i$$

→ **Azimuth.**
is an angular measurement in a spherical coordinate system. The vector from an observer (origin) to a point of interest is projected perpendicularly onto a reference plane; the angle between the projected vector and a reference vector on the reference plane is called the azimuth.



→ **Bijection.**

$$S, R \text{ are sets} \tag{1.1}$$
$$\forall i \in S, \exists! f(i) \in R \wedge \tag{1.2}$$
$$\forall i \in R, \exists! f(i) \in S \tag{1.3}$$
$$\tag{1.4}$$

→ **Binomial.**
A polynomial which is the sum of two terms.

→ **Boundary value problem.**
A differential equation with additional restrains. A solution to a boundary value problem is a solution to the differential equation which also satisfies the boundary conditions.

→ **Cauchy sequence.**
a sequence whose elements become arbitrarily close to each other as the sequence progresses. Think of *two* curves that approximate each other.

→ **Cardinal number, .**
The size of a set.

→ **Closure.**
The closure of a set is the points in the set including the limit points.

→ **Convergence.**
A series is convergent is the sum of it's elements approaches a given number.

→ **Convolution.**
Dictionary: a coil or twist, especially one of many.
Informal: an expression of how a shape from one function is modified by the other.

Can also be used to smoothen a discontinous a function (making it continous on the given range). We need to normalize our $g(x-\tau)$ so that we do not continously increase the $f(x)$

A convolution in time domain is equal to multiplication in frequency domain.

Convolution is a common way to combine wavelet to recreate signals.

→ **Continuous.**
In mathematics, a continuous function is a function for which "small" changes in the input result in "small" changes in the output.

As an example, consider the function $h(t)$, which describes the height of a growing flower at time t. This function is continuous. By contrast, if $M(t)$ denotes the amount of money in a bank account at time t, then the function jumps whenever money is deposited or withdrawn, so the function $M(t)$ is discontinuous.

→ **Continuous variables.**
There are three types of continuous variables:

Nominal Categorized within groups: box 1, 2, 3 or 4.

Dichotomous Boolean, yes or no.

Ordinal A nominal variable, just that different groups give different values. E.g. to like something on a scale of 1 to 10 gives you a ordinal range

→ **Conjugate.**
is a Binomial formed by negating the second term of a Binomial. The conjugate of $x+y$ is xy.

A functions' conjugate is often denoted $\overline{f(x)}$.

→ **Concave.**
Let f be a function defined on the interval $[x_1, x_2]$. This function is concave according to the definition if, for every pair of numbers a and b with $x_1 \leq a \leq x_2$ and $x_1 \leq b \leq x_2$, the line segment from $(a, f(a))$ to $(b, f(b))$ lies on or below the function.

- The sine function is concave on the interval $[0, \pi]$
- Concave if every line segment joining two point is never above the graph
- Concave functions has $f''(x) \leq 0$ in a given interval

→ **Congruence.**
Similar shape and growth, just different scalar / rotation.

→ **Damping.**
is an influence within or upon an oscillatory system that has the effect of reducing, restricting or preventing its oscillations. Underdamped systems will start bouncy and then stop.

→ **Differential operator.**
An operator to do differentiation. This is mainly to abstract differenentiation.

→ **Discrete Fourier Transform.**
converts a finite list of equally spaced samples of a function into the list of coefficients of a finite combination of complex sinusoids, ordered by their frequencies, that has those same sample values.

→ **Discrete Laplace transform.**
An integral transfrom, which in 2d uses the kernel

→ **Discrete Sine Transform.**
Similar to Discrete Fourier Transform, but uses only a real matrix.

→ **Discretization.**
concerns the process of transferring continuous models and equations into discrete counterparts. AKA smoothening curves to avoid jumps.

→ **Displacement.**
The shortest distance from source s to t (like air-distance).

→ **Divergence.**
measures the magnitude of a Vector Field's source or sink at a given point, in terms of a signed scalar.

E.g. air can be thought of to have a point s and t, where they push out hot and cool air, respectively. From these points, you can create a Vector Field that shows how air spreads from s and t. The divergence measures the collected value from each of these Vector Fields.

The operator for divergence is {div}, represented by nabla.

Given vector field $F = Ui, Vj, Wk$:

$$div\mathbf{F} = \nabla \cdot F = \frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} + \frac{\partial W}{\partial z} \tag{1.5}$$

→ **Elementary function.**
In mathematics, an elementary function is a function of one variable built from a finite number of exponentials, logarithms, constants, and nth roots through composition and combinations using the four elementary operations (+).

→ **Even function.**
Kind of like a mirror (90 degrees) around an axis;

$$f(x) = f(-x) \tag{1.6}$$

(1.7)

See also Odd function

→ **Expansion function.**
A function, with a series, to express a function. The series will in most cases be an approximation to the original function. Signal functions are often modeled using expansion functions.

→ **Eulers formula.**
$e^{ix} = \cos x + i \sin x$

→ **Explicit methods.**
In numerical analysis, an explicit method calculates the state of the system using only earlier history. See also Implicit method.

→ **Extreme point.**
A point furthest away from something.

→ **Field.**
A physcial quantity that has a value for each point in space and time.

→ **Finite difference.**
The difference in a function for $f(x+a) - f(x+b)$.

There are three types:

Forward is of the form $\triangle f(x) = f(x+h) - f(x)$

Backward is of the form $\nabla f(x) = f(x) - f(x-h)$

Centered is of the form $\delta f(x) = f(x + \frac{1}{2}h) - f(x - \frac{1}{2}h)$

→ **Finite element method.**
(FEM) is a numerical technique for finding approximate solutions to boundary value problems for differential equations. It uses variational methods (the calculus of variations) to minimize an error function and produce a stable solution. Analogous to the idea that connecting many tiny straight lines can approximate a larger circle, FEM encompasses all the methods for connecting many simple element equations over many small subdomains, named finite elements, to approximate a more complex equation over a larger domain.

→ **Filter bank.**
To take a signal and expose it to a series of filters that separate the input signal into different channel. One example is sound equalizers: you can adjust different parts of the signal.

→ **Fourier transform.**
Map one function onto another function expressed in terms of a series of sine and cosine terms, that when joined together form a translated domain.

Fourier transforms have the property that they can cover all the frequencies from the original input, but the mapped frequency loses the time domain: one can not determine at what time what frequency "stuff" was sent.

→ **Fourier Analysis.**
the study of the way general functions may be represented or approximated by sums of simpler trigonometric functions

→ **Flow.**
Motion of particles in a given set.

→ **Galerkin methods.**
Methods to convert continous operator problems to discrete ones.

→ **Generalized function.**
A distribution without steps, i.e. it is continious.

→ **Group.**
A set and an operator, such that if you take two elements from the set, apply the operator to them, then you will "receive" an element that could also be a part of the set (closure). Also it must satisfy for $\forall a, b \in S \rightarrow, (a \cdot b) \cdot c = a \cdot (b \cdot c)$ (associativity), ... also indentity and inverses. Example: integers and "+".

Groups have symmetry.

→ **Harmonic numbers.**
$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + 1n = \sum_{k=1}^{k} \frac{1}{k} \simeq \ln n$

→ **Heavyside Step Function.**

$$H(x) = \begin{cases} 0 & \text{for} x < 0 \\ \frac{1}{2} & x = 0 \\ 1 & \text{for} x > 0 \end{cases}$$

(it looks just like you think...)

→ **Heisenberg's uncertanty principle.**
kjkj

→ **Homogenous.**
If a function f is multiplied by a sclar k, then the result is a number multiplied by the scalar raised to some power a.

→ **Hyperbolic.**
Two curves that kind of face each other like bananas.

→ **Identity function.**
$\forall x, f(x) = x$

→ **Implicit method.**
In numerical analysis, an implicit method calculates the state of the system using both the current state and future ones. E.g. Gauss-seidel method See also Explicit methods.

→ **Idempotence.**
An operation that can be applied multiple times without changing the the result. E.g. $f(f(x)) = f(x)$

→ **Idenpendent variable.**
A variable that may be given without considering the value of other variables. E.g. for $y = 4x + 2$, x is indepdent (can be chosen freely) whereas y is not: it depends on the value of x.

→ **Invariant.**
A sentence that is believed to be true throughout some timespan.

→ **Integral.**
The "reverse" to a derivate.

$$\int_a^b x^n dx = \frac{x^{n+1}}{n+1} + C \tag{1.8}$$

(1.9)

A function is said to be *locally integrable* if it's integral is finite in a given domain.
Integration by parts: $\int u \times v dx = u \int v dx - \int u\prime(vdx)dx$
Some identities:

$$\left| \begin{array}{l|l} \int_a^b \cos x & \sin x + C \\ \int_a^b \sin x & -1 \times \cos x + C \\ e^x & e^x + C \end{array} \right|$$

→ **Integral transform.**
Input a function, and output another (using integrals). To do the transformation, one typically usues a *kernel function* (e.g. using a stencil in a grid).

Usually, this is done to find a domain or function that is easier to compute on.

After completing the computations, one performs an inverse integral transform to translate back to the original domain.

→ **Interpolating polynomial.**
Given a set of points, define a function f that intersects these points.

→ **Kronecker delta.**
A function for two variables, that returns 1 if the variables are equal, and zero otherwise.

$$\delta_{i,j} = \begin{array}{ll} 0 & if\, i \neq j \\ 1 & if\, i = j \end{array}$$

→ **Law of cosines.**

$$v \cdot u = |v||u| \cos\theta$$

Note that if v, u are unit vectors, their lengths are both 1. We can then rewrite the expression as

$$v \cdot u = \cos\theta$$
$$\arccos(v \cdot u) = \theta$$

→ **Laplace operator.**
Transform a function of f of t to a function f of s
A differential operator given by the Divergence of a gradient function in euclidian space.

→ **Laplacian Matrix.**
sometimes called admittance matrix, Kirchhoff matrix or discrete Laplacian, is a matrix representation of a graph.

→ **Lattice.**
In mathematics, especially in geometry and group theory, a lattice in $\mathbf{R}^n$ is a discrete subgroup of $\mathbf{R}^n$ which spans the real vector space $\mathbf{R}^n$. Every lattice in $\mathbf{R}^n$ can be generated from a basis for the vector space by forming all linear combinations with integer coefficients.

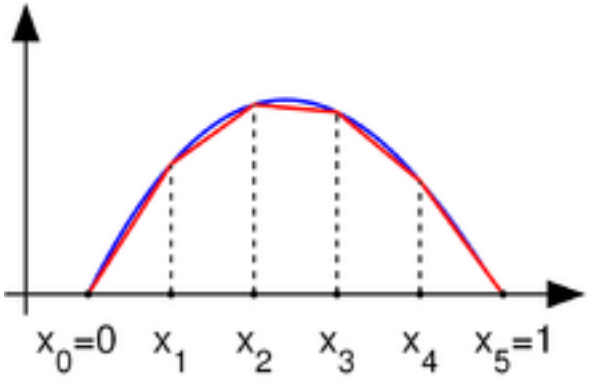
→ **Line integral.**
An integral where the function to be integrated is along a curve. The function is usually a Vector Field or Scalar field.

→ **Locus.**
a set of points whose location satisfies or is determined by one or more specified conditions

→ **Merit function.**
A value that says something about how good we have acheived a goal.

→ **Moment.**
TBD.

→ **Numerical analysis.**
The study of algorithms that use numerical approximation (as opposed to general symbolic manipulations) for the problems of mathematical analysis.
Given a space V, find a finite subspace of V, such that you can calculate on it...



A function in with zero values at the endpoints (blue), and a piecewise linear approximation (red)

→ **Odd function.**
A function with the same values when you rotate 180 degrees around …

$$-f(x) = f(-x) \tag{1.10}$$
$$f(x) + f(-x) = 0 \tag{1.11}$$

See also Even function

→ **Pathological.**
Something that is counter-intuitive. The opposite is **well-behaved**.

→ **Parabola.**
A U-shaped, 2d, symmetrical curve.

→ **Parameterization.**
Represent a curve as a function.

$$x = \cos t \tag{1.12}$$
$$y = \sin t \tag{1.13}$$

is the parametric representation of a unit circle.

→ **Partial derivative.**
Given a function f with multiple parameters, a partial derivative is a derivative with respect to one of those variables. Partial derivatives are often denoted by ∂.

To use partial derivation, you often assume that the other variables are constants. Otherwise, there is an infinite number of tangent lines at any point, so you will have to have some range on the tangents.

→ **Partial differential equation.**
A set of variables, and equations that show how they are all linked together.

→ **Pertubation theory.**
The "art" of finding an approximate solution to a problem by starting with an exact solution to an easier problem.

→ **Power series.**

$$f(x) = \sum_{n=0}^{\infty} a_n (x-c)^n \tag{1.14}$$

→ **Rectangle function.**

$$\Pi(x) = \begin{cases} 0 & \text{for} x > \frac{1}{2} \\ \frac{1}{2} & \text{for} x = \frac{1}{2} \\ 1 & \text{for} x < \frac{1}{2} \end{cases}$$

→ **Reference value.**
The optimal/starting value. Note that when calculating differences, a reference value implies that order matters, i.e. $x - y \neq y - x$.

→ **Residual.**
In numerical analysis, this is the error from a result. E.g. in integration we have something-something + C, where C is the the residual.

It can (perhaps betterly) be stated that residuals is the error that occurs when approximating a function.

→ **Riemanns sum.**
Sum from an integral. Divide the area under/over a curve into rectangles or trapezoids, and sum together their area. The smaller the shapes, the better.

It is important to note that Riemanns sum is an approximation - one can not perfectly reconstruct the area under a curve.

→ **Risk function.**
Gives the expected value from a lossy function (compare two results, diff).

→ **Sine wave.**
or sinusoid is a mathematical curve that describes a smooth repetitive oscillation.

Its most basic form as a function of time (t) is:
$y(t) = A\sin(2\pi f t + \varphi) = A\sin(\omega t + \varphi)$ where:

A , the amplitude, is the peak deviation of the function from zero.

f , the ordinary frequency, is the number of oscillations (cycles) that occur each second of time.

ÏĽ $= 2$ÏĂf, the angular frequency, is the rate of change of the function argument in units of radians per second

φ , the phase, specifies (in radians) where in its cycle the oscillation is at t = 0. I.e. the "horizontal shift" from a regular sine wave(????). If you e.g. represent two functions, $f(x) = \sin x$ and $g(x) = \sin x + 3$, the difference, i.e. phase shift, is 3. Note that these two functions have the same amplitude and frequency.

→ **Standard form.**
To write a number as a power of 10.

→ **Stiffness matrix.**
A system of linear equations that must be solved to get an approximate solution to a differential equation.

To do this, consider Poission problem:

→ **Taylor series.**
A representation of a function as an infinite sum of terms that are calculated from the derivatives at a point in the function. The formula is:

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \tag{1.15}$$

→ **Shift invariant system.**
A discrete form of Time invariant system - the timesteps are discretized to follow a lattice spacing.

→ **Stationary point.**
A place in which the derivative is zero.

→ **Symmetric function.**
A function where the ordering of the parameters does not matter.

$$f(x,y) = x + y \tag{1.16}$$
$$f(y,x) \equiv f(x,y) \tag{1.17}$$

→ **Time invariant system.**
A function with an output that does not depend on the output:

$$x(t) = y(t) \tag{1.18}$$
$$x(t+\delta) = y(t+\delta) \tag{1.19}$$

One example is the derivative function of polynomials (and others?) - no matter what timestep you choose - the value if the same.

The "antonym" would be a time variant system.

→ **Translation invariant system.**
A system such that after a translation from A to T, any operator applied to A yields the same results in T.

→ **Unity.**
The number one. "Unit functions", "unit circles" all have the number 1.

→ **Univariate.**
An equation with only one variable.

→ **Vector Field.**
An assigment of direction for a given set of points in an euclidian space. E.g. select every (10n, 10n) pixels in an image and get their derivative.

→ **Weak formulation.**
Transfer a concept from linear algebra to solve equations in other field.

Equations that have weak formulations are not required to be correct for all input (:: weak solutions, PDE might not have derivates but you make pretend)

→ **Window function.**
A function that is zero outside of a given domain.

1.3 Graphs

→ **Arborescence.**
an arborescence is a directed graph in which, for a vertex u called the root and any other vertex v, there is exactly one directed path from u to v.

→ **Clique.**
A subset of vertices $C \subset V$, such that in this subgraph all nodes are connected, i.e. there is an edge from every pair of nodes.

→ **Connected graph (component).**
A graph in which, from any $v \in V$, you can reach any other $u \neq v$

→ **Directed graph.**
Properties:

- $\sum_{v \in V} d(v) = 2|E|$
- $\sum_{v \in V} indegree(v) = \sum_{v \in V} outdegree(v)$

→ **Eularian.**
Visit each *edge* once.

→ **Hamiltonian.**
Visit each *vertex* once.

→ **Metric spaces.**
See also Semi-metric Properties:

- $d_{u,v} = 0 \iff u = v$
- $d_{u,v} = d_{v,u}$
- $\forall k, d_{u,v} \leq d_{u,k} + d_{k,v}$

→ **Minimum mean cost.**
minimize ratio of cost of arcs (directed edges) to number of arcs

→ **MST.**
A mininum Spanning tree, such the weight of this tree is less than or equal to all other possible Spanning tree.

→ **Planar graph.**
No edges need to cross each other.

→ **Semi-metric.**
Like a Metric spaces, without the property that $d_{u,v} = 0 \iff u = v$

→ **Spanning tree.**
Connected, undriected graph that has all vertices and some subset of edges to form a tree.

→ **Strongly connected component.**
Similar to Connected graph (component), but in a directed graph.

→ **Topological sort.**
Runs in O(V + E), same as DFS, although it can be supered to $O(\log_2 n)$

→ **Tree.**
Properties of a tree:

- in a tree, there will always be an even amount of nodes that has an uneven degree.
- Degree of nodes in a tree is at most twice the number of nodes

→ **Vertex-cover.**
A selection of vertices such that each edge is incident to at least one of them.

1.4 Image Processing

→ **Aliasing.**
Images are constructed and then reconstructed one or more times - each of these reconstructions are referred to as aliases.

→ **Anisotropy.**
Directionally dependent. Opposite of Isotropy. Example is to apply a filter to a set of images - consider what would happen if one image was tilted.

→ **Anisotropic diffusion.**
also called Perona-Malik diffusion, is a technique aiming at reducing image noise without removing significant parts of the image content, typically edges, lines or other details that are important for the interpretation of the image.

→ **Aperture.**
A hole which light goes through.

→ **Convolution.**
In image-processing, this can in general be thought of as applying a mask that takes the local neighbourhood for a pixel x and calculates its gradient.
Noteworthingly, convolution as an operation to an image is in general $O(n^2)$.

→ **Curve representation.**
How to represent a curve: computational cost is crucial.

Explicit $y = f(x)$ - only lines but really easy to genereate.

Implicit $f(x,y) = 0$. ...imagine a circle: to know if a point is on the line, we can use pythagoras's $x^2 + y^2 - r^2 = 0$.
Costs a lot.

Parametric $(x,y) = (f(u), g(u))$. Oh yeah..

→ **Dilation (of functions).**
To skew a function, e.g. for $f(x) = x$, you skew to $\hat{f}(x) = x + 2$.

→ **Dirichet boundary condtions.**
In image processing, you can say that this boundary condition is to assume for a set of differential equations, the unknowns at the borders are 0: $u(0) = u(1) = 0$.

→ **extrapolation.**
The process of estimating, beyond the original observation range, the value of a variable on the basis of its relationship with another variable.
Creating a tangent line at the end of the known data and extending it beyond that limit.

→ **Filter.**
Applying a mask or operation to an image to extract or distort values.

High-pass takes only the high frequency into play, lower values are not affected that strongly (if at all)

Low-pass ...

→ **Gaussian blur.**
Use a gaussian function on an image to reduce image noise and detail

→ **Gaussian filter.**
Gaussian filters have the properties of having no overshoot to a step function input while minimizing the rise and fall time.

→ **Gaussian pyramid.**
A stack of images, where for each iteration, you take neighboring pixels and make them into one. That is, you reduce the intensity in the picture. This compresses the image.

It is a pyramid because for each iteration you take neighboring pixels from the previous pyramid to generate the current.

→ **Gradient flow.**

$$V = \nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right)$$

Note that for an oridnary image, the gradient at a given position is given by:

$$\nabla f = \frac{\partial f}{\partial x}\hat{x} + \frac{\partial f}{\partial y}\hat{y}$$

Where the first (x) term is the gradient in direction x, and y is the gradient in direction y.

One can also calculate the direction of the gradient using $\theta = a\tan^2 \frac{\partial f}{\partial y}, \frac{\partial f}{\partial x}$

→ **Grating.**
A collection of identical pararell objects, placed next to each other with equal spacing.

→ **Grayscale.**
an image in which the value of each pixel is a single sample, that is, it carries only intensity information

Grayscale images are distinct from one-bit bi-tonal black-and-white images, which in the context of computer imaging are images with only the two colors, black, and white (also called bilevel or binary images). Grayscale images have many shades of gray in between.

Grayscale images are often the result of measuring the intensity of light at each pixel in a single band of the electromagnetic spectrum

→ **Image gradient.**
Gradual blend of color

→ **Image noise.**
Unwanted signal, electrical signals not wanted in an image

→ **Image masking.**
Apply an image over another. This can be used for e.g. putting a square image with only contents in the middle over another.

→ **Interpolation.**
A way to estimate (pixel) values when resizing to a larger image.

An example from calculus is: given temps at noon and midnight, estimate the temp at 5 PM as the mean of noon and midnight.

Interpolation can also (in signal processing) be referred to as **upsampling**.

Interpolation can be *linear*(1D), *bilinear*(2D) and so on...

→ **Isotropy.**
Directionally independent. Opposite of Anisotropy.

→ **Laplacian.**
Highlight regions of rapid intensity change and is therefore often used for edge detection.
Often applied to an image that has first been smoothed with something approxiating in order to reduce its sensitivity to noise.

→ **Spatial frequency.**
The number of changes in color values that occur per space.
Images with high spatial frequency are detailed, images with low spatial frequency will appear blurred.
I.e. sharp transitions from low/high to high/low intensities are said to have a large spatial frequency,

→ **SRGB.**
sRGB is a standard RGB color space created cooperatively by HP and Microsoft in 1996 for use on monitors, printers and the Internet.

→ **Poisson image editing.**
Blend two images together and make them seem alike.

→ **Raster order.**
Begin at top left, proceed to right, then at leftmost pixel in next line.

→ **Raster image.**
An image with pixels.

→ **Ringing.**
The size of the oscillations after a peak frequency. E.g. if you have a sudden overshoot (bright color) the amplitude of the next wave is the ringing. (???)

→ **Scalar field.**
Associate a value to every point in a space. E.g. for an image, you could assign each pixel a color value.

→ **Seperable filter.**
A filter that can be written as the product of two or more small filters. This is usually done to reduce the computational costs. E.g. Convolution is usually faster in 1d than iterating nD, for $n \geq 2$.

→ **Spline.**
A line that goes through a series of points.
The section between the lines is interpolated. This can be done linearly, quadratic, ... so on.

→ **Stencil.**
A figure that connects dots on a grid. Usually a cross-like figure (one center dots, and one edge out in 4 directions from the center to other dots).

1.5 Linear Algebra

→ **Basis.**
Given a set of vectors in $\mathbf{R}^n, V$, which is linearly independent, the set V is a *basis* if you can span $\mathbf{R}^n$ using V.

→ **Backward substituion.**
Opposite of Forward substituion.

→ **Condition number.**
A measure of how the output value for a function changes respective to input variables.

→ **Column space.**
All linear combinations of the columns of a matrix A.

→ **Consistent.**
$\iff$ the rightmost column of an augmented matrix is not a pivot column. I.e. in a 3x4 matrix, if the last row is zero and 2nd column has pivot column, then the system is still consistent.

→ **Diagonal Dominance.**
$\forall i, \exists i A_{i,i}, \forall i, \iff A_{i,i} \geq \sum_{j=0, j\neq i}^{m}$, then matrix A is DD.

→ **Diagonlizable matrix.**
Given a matrix A, we can express it as $A = PDP^{-1}$. Then, we can compute:

$$A^2 = (PDP^{-1})(PDP^{-1}) = PD(P^{-1}P)DP^{-1} = PDDP^{-1} \tag{1.20}$$

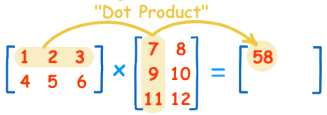
$$\text{Because} P \cdot P{-}1 = I \tag{1.21}$$

$$\ldots A^k = PD^kP^{-1} \tag{1.22}$$

So, we use diagonal matrices to easily raise matrices to power k.
To diagonalize a matrix $A^{n\times n}$, it is required that it has n linearly independent eigenvectors.

→ **Divergence.**
In vector calculus, divergence measures the magnitude to the gradient of Vector Fields at a given point.

→ **Dotting matrices.**



→ **Echelon form.**
A matrix where:

1. All nonzero rows are above zero-rows
2. Each pivot column is placed from left to right

→ **Eigenvector.**
Given a square matrix A, when A is multiplied with an eigenvector v, the resulting matrix A′ is a multiple of v. The multipe is denoted by λ and is called an eigenvalue. So, $Av = \lambda v$

→ **Forward substituion.**
In an iterative method, if we first solve an element $A_{i,j}$, then when we caluclate some $A_{i+a,j+b}$ in the same iteration, then we use the updated value of $A_{i,j}$. Opposite of Backward substituion

→ **Gaussian Elimination.**
AKA "Row reduction". Add row x to row $y, y \neq x$, to reduce y to zeroes.

→ **Ill-conditioned matrix.**
A matrix is ill-conditioned if the Condition number is large. If you make a small change in an ill conditioned matrix, there are usually large differences in results from calculations with this maxtrix.

→ **Inner product.**
The product $u^T v$. Given that $u, v \in R^{i \times j}$, then we need to transpose to perform normal dot products.
Inner products are commutative: $u\dot{v} \equiv v\dot{u}$.

→ **Inverse.**
For a matrix $A \in \mathbf{R}^{2x2}$:

$$A^{-1} = \frac{1}{|A|}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix} = \frac{1}{ad - bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

For a matrix $A \in \mathbf{R}^{3x3}$: bcacab
Properties:

- $(A^{-1})^{-1} = A$

→ **Kernel.**
For a vector space given by a Linear transformation, a kernel is the set of vectors $v \in V$, s.t. $T(u) = 0$.

→ **Length of Vector.**
For a vector

$$v = [a_1, a_2, \ldots, a_n]$$
$$|v| = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}$$

→ **Linear dependence.**
For $V = v_1, \ldots, v_n$, and $\forall v, visvector$, if none of the vectors in V can be written as a linear combination from the other vectors in V, the set is linearly independent.

→ **Linear transformation.**
Take a vector space into another, s.t.

$$T(u + v) = T(u) + T(v) \forall u, v \in v \tag{1.23}$$
$$T(cu) = cT(u) \forall u \in V \tag{1.24}$$

→ **Matrix of observations.**
Collect a sample, subject it to tests, and for each test, you give a value. For n tests and m samples, you will result in $\mathfrak{Re}^{n \times m}$ *matrix of observations*.

→ **Norm.**
The "length" of a vector, just remember to sum up over all dimensions. Each number is raised to n, and the total sum is then raised to

L1

L2 Can be defined as:

$$\phi^2 = \phi \times \phi = \cdots = \int \phi(x)^2 dx \tag{1.25}$$

$$|x| = \sqrt{\sum_{k=1}^{n} x_k|^2} \tag{1.26}$$

$\frac{1}{p}$.

→ **Normalization of vectors.**
$\hat{X} \equiv \frac{X}{|X|}$, where $|x|$ is the Length of Vector X is, in this case, evaluated as the additive sum of it's entries.

→ **Null space.**
Given a matrix A, if you solve for that each row = 0, all possible values for each x makes out the Null space.
Note that here it is apossible to get free variables for some x, and bound to others.

→ **Numerical stability.**
This concept describes how changes in the input should not affect the result. An example could be that sorting a list should not change the outcome from applying a function to the input.

→ **Orthogonal.**
Two lines that intersect each other at 90 degrees.

- Orthogonal matrices preserve dot products: given two vectors u, and v, and an orthogonal matrix Q, the following is true: $u \times v = Qu \times Qv$
- The determinant of an orthogonal matrix is always 1 or -1
- The transpose of Q is equal to it's inverse, hence: $Q \times Q^T = I$

→ **Orthonormal.**
In Inner product space are orthonormal if they are orthogonal and unit vectors.

→ **Perpendicular.**
Similar to Orthogonal, but with lines.

→ **Pivoting.**
Finding the first non-zero element in an algorithm on linear systems. To do so, you can do things like gaussian elimination, etc.

Complete pivoting find the largest absolute value of a pivot, considering all elements.

Partial pivoting finds the largest absolute value in the pivot column.

Scaled pivoting finds the largest absolute value of a pivot column relative to it's entries in the same row.

→ **Plane.**
A flat, two-dimensional surface

→ **Positive semidefninite matrix.**
Properties:

- Nonnegative Eigenvectors
- $X = V^T V$ for some $V \in \mathbf{R}^{mxn}$
- $X = \sum_{i=1}^{m} \lambda_i w_i w_i^t$ for some $\lambda_i \geq 0$ and vectors $w_i \in \mathbf{R}^n$ such that $w_i^T w = 1$ and $w_i^T w_j = 0$

→ **Preconditoner.**
A matrix P such that $P^{-1}A$ has a smaller Condition number than A.

→ **Projection.**
define a vector v and u.

$$L = \{cv | c \in \mathbf{R}\}$$
$$proj(v) = l \in L \text{ such that } u - proj(v) \text{ is Orthogonal to l}$$
$$\textit{I.e., } proj(v) = cv, c \in \mathbf{R}$$

Properties:

- $proj(v) = proj(v)^2$
- Linear independence on u, v also relates for $v - proj(v), u$
- adding $proj(v)$ to v gives you u

→ **Relaxation methods.**
relaxation methods are iterative methods for solving systems of equations, including nonlinear systems. Examples are Gauss-Seidel, Jacboi, etc.

→ **Singular matrix.**
Any square matrix without an inverse. A matrix is singular $\iff$ its determinant is 0.

→ **Span of vectors.**
All linear combinations of a set of vectors.

$$V = \{v_1, \ldots, v_n\}$$
$$C = \{c \in C | \mathbf{R}\}$$
$$span = c_1 v_1 + \cdots + c_n v_n$$

→ **Spectrum of a matrix.**
The multiset of its eigenvalues

→ **Spectral Radius.**
The largest absolute eigenvalue of a matrix.

→ **Successive Over Relaxation.**
Gauss-Seidel is the same as SOR (successive over-relaxation) with $\omega = 1$

→ **Symmetric.**
- Length of rows is equal
- The transpose is equal to the originl

→ **Tensor.**
Geometric objects that describe linear relations between vectors or scalars.

→ **Toeplitz matrix.**

$$\begin{bmatrix} a & b & c & d \\ b & a & b & c \\ c & b & a & b \\ d & c & b & a \end{bmatrix}$$

Noteworthingly, this kind of matrix can be inverted in $O(n \times \log n)$

→ **Trace.**
Sum of all diagonal entries in a matrix. $A_{11} + A_{22} + A_{nn}$

→ **Transpose.**
Take column i and make it into a column. Repeat.

→ **Unit vector.**
A vector who's length is 1.

→ **vector length.**
number of "steps" in a vector

1.6 Misc

→ **Continuum.**
Continuum theories or models explain variation as involving gradual quantitative transitions without abrupt changes or discontinuities

It can also be the set of real numbers, or

→ **Here document.**
here document (here-document, heredoc, hereis, here-string or here-script) is a file literal or input stream literal.

When used simply for string literals, the ¡¡ does not indicate indirection, but is simply a starting delimiter convention.
Example: (bash)

```
tr a-z A-Z << END_TEXT
one two three uno dos tres
END_TEXT
```

→ **Information space.**
is the set of concepts and relations among them held by an information system; it describes the range of possible values or meanings an entity can have under the given rules and circumstances.

1.7 Programming Security

→ **Accessory controls.**
If a company X notices that a part P of a product does not belong to them, they may reduce the effectiveness of the product.

For example, if a printer notices a competitors ink in the printer, it may go from high quality to low.

This is considered a form of authentication.

→ **Access control.**
I assume the meaning is intuitive to understand. What's worth mentioning is that access control can operate on four levels, namely:

- Application
- Middleware
- Operating system
- Hardware

→ **Access Control Lists.**
Most UNIX-fans will know them already: *-rw-r–r– 1 <owner of file> <group owner of file>*. This is an example of a rowwise mandatory ACL- where an object has a given set of definitions for three different groups: the first three letters in the string denotes what the owner can do (in this case, there is no execution, but read and write), the third next letters (mid-block) denote what members of the owner group can do to the file, and the next three letters denote what everyone else can do.

This type of ACL are slow for systems with many users. The vantage is that is efficient. Other forms of ACL's include database access control (suffering from inability to model states, etc.

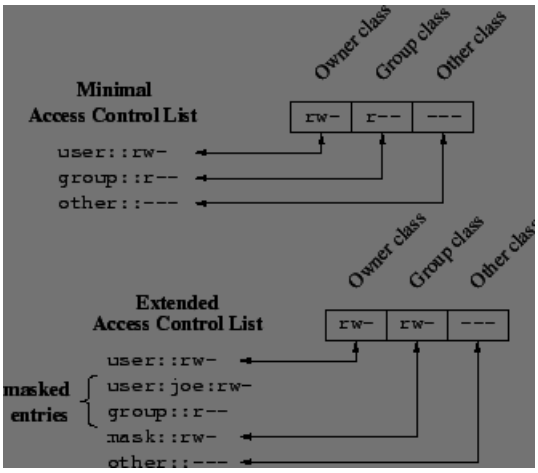


Figure 1.1: Typical UNIX ACL

→ **Assurance classes.**

→ **Authentication.**
To prove that you are who you claim to be. E.g. to provide a password for an account, etc.

→ **Authorization.**
To ensure that a user has sufficient permissions to do what he/she asks for. E.g. an authenticated user may log in, but may not change the passwords for other users, because this authenticated user is not authorized for this account.

→ **BAN-logic.**
IS THIS REALLY NECESSARY?
...formal reasoning in cryptology, somewhat alike first-order-logic.

→ **Biba Model.**
Essentially BLP, just that it is read by "no read down, no write up". The BLP can be categorized as "no write down, no read up".

> *A monk may write a prayer book that can be read by commoners, but not one to be read by a high priest*
>
> — Wikipedia

→ **Bell LaPadula model.**
Any system that adheres to the two following properties, are said to adhere to the Bell LaPadula Model, which is a Security Policy (model):

Simple security property *no process may read data at a higher level.* If a process is on runlevel X, it should not have access to data at level X + 1.

***-property** *no process may write data to a lower level.* Imagine that a sysadmin gets a Trojan that reads much sensitive data. The *-property prevents this data to be written down to a lower level where it is accessible to other accounts with lower clearance.

It is worthwile noting that the properties fail to mention policies for creation and destruction of files. Furthermore, the model is broken if sensitive data is temporarily declassified. The data can then be manipulated without violating the constraints of BLP. Hence, to make the BLP safe, we need to introduce the **tranquility property**:

Strong: Security labels on objects never change during systems operation.

Weak: Security labels never change in such a way that it defines the given security policy.

However, the problem with the tranquility property is that process will have a difficult time reading files. If at first one reads something at a high level, then no concurrent read/write operation can occur for a lower level. Applications will therefore need to be customized extensively to accomodate for the tranquility property.

The BLP only deals with confidentiality, not integrity (see Biba Model)

→ **Biba Model.**
Deals with integrity alone, ignores confidentiality.

→ **BMA model.**
British Medical Association. Basically a 9-point long list wich comprehends the BLP, but also represents state, and so on. Used for medical records.

It's worthwhile noting that for medical records, it's much harder to define privacy and such because doctors are often required to involve third parties with their records. This could be researchers, drug companies, family, so on.

→ **Capture error.**
People are used to clicking "OK" on alert boxes without reading, so you might fool them intro clicking on one of your buttons by simply making a popup box.

→ **Chinese Wall.**
Lets say that you work in finance, reviewing different oil companies. If you've worked on account A for some time, you might have information that is relevant for account B. To prohibit this, you're given a timed restriction, such that you cannot work on other accounts if you have sensitive information.

The general idea is to say that "you can access this, but nothing else, there's a chinese wall between you and other accounts."

The chinese wall has been expressed to be similar to the BLP. Let c denote objects of interest, for example bank data. s denotes a subject, for example a person, who tries to access objects. $y(x)$ is a function y for an object x that denotes a company y's interest for the object x. Similarily, for a company f, there's a function $f(x)$. The domain $S = S(s)$ is a set which covers all objects that s can access.

> **Simple security property** $\forall s$, s will have access to c iff $\forall o \in S(s)$, $y(c) \cap x(o) = \emptyset$ or $y(c) \in y(o)$
>
> **The *-property** s can write to an object c iff $\forall o\{m \| m \in S(s)\}), o \notin x(c)$ and $y(c) \neq y(o)^{[what?]}$

→ **Common Criteria.**
Defines many Security functional requirements. ISO-standard.

→ **Confidentiality.**
Who/what could have read this message?

→ **Covert channel.**
If a system adheres to the BLP, a low security object might still commuicate with one at a higher level. If the two share a resource, e.g. a disc, one could make the higher level process do something with the disc head to communicate. This could for example be invoking an error at time t_i to indicate that bit i in an important file is either 1 or 0. This way of communication is called a covert channel.

→ **Cryptographic nonce.**
a nonce is an arbitrary number used only once in a cryptographic communication. The nonce is there to ensure freshness of a message, e.g. it could be a timestamp. That way, one can be assured that encrypted messages are recent and not old.

→ **CSRF.**
performs an action on the server. The user is typically not aware of what he/she is doing. I.e., if there is a URL that can be used to purchase 50 cars, Eve can send that to Alice and make her, unwillingly, buy 50 cars.

CSRF attack can be mitigated by properly implementing session authentication tokens, such that one cannot modify/use an account without having a valid token from the server. That way, the POST to purchase cannot be instantiated unless the client performs a series of steps.

→ **Discretionary Access Control.**
A simple form of access control. The ease of implementing DAC makes it popular. DAC is similar to Mandatory Access Control, except from that there are no *policies* for objects. This means that whenever a subject wants to apply an action to an object o, the operating system will only evaluate s and o, not the action that s wants to apply, in order to determine whether or not s's action will be executed.

→ **Difference between DAC and MAC.**
DAC is not a multilevel security protocol in that, for any subject s, **any** operation on an object o is either granted or not **only by considering the relationship between s and o, not the action applied**.

Furtherly, the following quotes quite sum it up:

Systems can be said to implement both MAC and DAC simultaneously, where DAC refers to one category of access controls that subjects can transfer among each other, and MAC refers to a second category of access controls that imposes constraints upon the first.

— Wikipedia

In general, when systems enforce a security policy independetly of user actions, they are described as having madatory access control, as opposed to the discretionary access control in systems like Unix where users can take their own access decision about their files

— Page 246 in the book

→ **Evaluation Assurance Level.**
The Common Criteria uses

→ **Format string vulnerability.**
Input data will be used to format output. However, the input data goes to the stack, and there are therefore vulnerabilities.

→ **High water mark principle.**
Start off at the bottom and elevate permissions as you need them. E.g.às one opens a mail client, one finds

→ **Identify Friend or Foe.**
IFF: A system that can tell whether you are a friend of a foe. A classical crypto-problem; how can I know that I am not talking to Eve?

→ **Integer manipulation attack.**
Causing an under- or overflow or truncation such that you can exploit software.

→ **Integrity.**
Who/what could have altered this package?

→ **Kernel bloat.**
E.g. in Windows, you need to have many drivers run as root in the kernel. This will naturally open up for more security holes.

→ **Key-distribution.**
One entity on a network wants to talk to another in a network. We will assume Alice wants to talk to Bob. Her concern is that they might establish a connection, but Eve might hijack it and pretend to be either party. Alice needs to know that she is speaking with Bob, and she needs to know that Bob is not communicating with any Eve.

To prevent this, a trusted third party is introduced. We will call this Sam. We will assume that the connection to Sam is safe, and that Sam knows every user of the network. Alice contacts Sam. Her request is as follows: "I am Alice, and want to communicate with Bob". Sam returns two tokens. The first token can only be read by Alice, and the second only by Bob. Alice then contacts Bob, giving him only the second certificate. Bob decrypts the certificate, which unravels an encryption/decryption key. Bob and Alice can now encrypt and decrypt messages to each other using this key, and communicate without Sam.

→ **Landing pad.**
A piece of code, such that when exectued, the processor will exectue malicious code.

→ **Lattice model.**
Essentially equivalent to the BLP. You use labels such as "TOP SECRET", "SECRET" and say that there are no communications between levels as in the BLP.

You combine things, so for each item, e.g. for "missiles" you might have a clearance for something like "SECRET". But you also need codewords, such as "SECRET MISSILES" if you want to read the secret stuff about missiles.

The point of the lattice model is that you have aggregated values, such that a lattice will yield a numeric d. To get access to d you need $e > d$.

→ **Malware.**

→ **Mandatory Access Control.**
Also known as multilevel security. MAC is enforced as follows: for each object o, define a set of actions that are applicable to this object, and categorize them by access level. E.g. one could define deletions as *administrator-level* and modifications as *user-level*. The categorization of actions to objects is known as defining the **policy**. The policy is often stored as an Access Control Lists. Typical values for o include files, directories, ports, etc. When subject s requests an action a on o, the operating system looks up the rules for o, finds a, and evaluates s against the policy for a on o.

To clarify: in MAC, there is no such thing as "super-user-access", i.e. being a user with high privileges won't necessarily grant you privileges on all objects, because some objects might have policies that only permit certain other users to manipulate them. For example, even though you might be a system administrator, you should not be able to read other user's passwords, nor modify them. You might rather grant sysadmins the privilege to reset the password, and let authenticated users modify their own passwords.

MAC ensures a security policy indepent of user actions. That is, even though a user owns a file, he/she might not be able to do whatever he/she wants with it.

A good example of the usefulness of MAC is that a computer that is hacked may not jepordize other systems. Despite of having obtained some level of control, the attacker may not get access to other critical features.

MLS's are usually difficult to implement. One example is the **cascading** problem; two systems A and B may both have access to an object o. Simeltaneously, system A have access to another object a, system B has access to object b, where $a \neq b$. Now, modifying o might affect how a, b are treated. So if system B wants to alter a, it may modify o, then system A reads o, then modifies a.

Do also note figure 1.7 as it highlights a security caveat of MLS; low-level data can also be worthwhile protecting. See also the quote in Difference between DAC and MAC

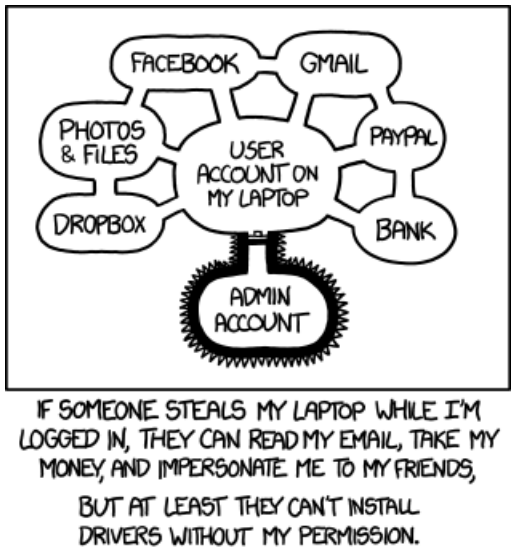


Figure 1.2: xkcd.com/1200

→ **No-operation.**
NO-OP: From assembly, do nothing.

→ **Phishing.**
Essentially the same as Pretexting, just that you're trying to decieve customers instead of staff.

→ **Pretext.**
To create a plausible scenario that can decieve personell with sensitive information to disclose this to Eve.

→ **Principle of least privilege.**
Programs should only have as much privilege as they need. There's no need to "sudo webbrowser"

→ **Protection profile.**
A document that identifies the desired security properties of a product. This is structured as a list of security requirements defined in a way that adheres to Common Criteria

→ **Pump.**
A one way transfer (data diode) that takes data from a low access level to a higher access level.

→ **Race condition.**
Process A reads resource X, process B reads resource X, process A writes, then process B writes. The value is bad, since B's original value was bad.

→ **Reflection attacks.**
First, see Challenge-Response. Suppose that the challenge-response is similar for two entities. I.e. party A sends a challenge N to someone. To solve the challenge, party E simply returns N to A, has party A solve it, and voila, party E knows the solution to N.

→ **Replay attack.**
Assume that to login, Alice encrypts her password with one million bits , three times, and sends her password to a login page. The login decrypts and accepts Alice's authentication token.

Eve could simply sniff the POST from Alice and the replay it to the server. Even though the password is highly encrypted, Eve's POST will be valid. The login page accepts Eve.

To mitigate, one should/could implement session tokens; see Session token. As mentioned in that section, however, session tokens also introduce other risks.

→ **Repudiation attack.**
To give someone a bad reputation. This could be by rating items with low scores, writing mean reviews, etc.

→ **Role based access control.**
Permissions are not related to users, but rather their functions. This means that if a sysadmin is sick, a second-in-rank user could assume the rank of the sysadmin to do whatever is necessary.

→ **Sandbox.**
A limited environment. It is common to host websites in sandboxes, so that even if the webserver is hacked, a hacker can't do much on the mainframe.

→ **Security Policy (model).**

→ **Secrecy.**

→ **Security assurance requirements.**

→ **Security functional requirements.**

> *A document that expresses clearly and consicely what the protection mechanisms are to acheieve. ... It will often take the form of statements about which users may access which data*
>
> *— page 240*

As with most computer policies, it's important the statements are succint, i.e. briefly and clearly expressed, without implications. Also, it's important to make sure that you explicitly define:

- Who determines the policy?
- What qualifies for "need to know"?
- How will the policy be enforced?

→ **Security requrement.**
A statement which defines what level of security is utilized for different kind of attacks. It's important that the requirements do not discuss design, only what is required (hence, requirement).

All requirements should be testable. Good ways to do this is to quantify. Quantifying security is difficult, however.

→ **Security target.**
A document that describes what a product does, or at the very what it does that has an impact/relevance in security contexts.

→ **Session token.**
Also known as session ID or session identifier. A session token is a unique identifier, usually in the form of a hash generated by a hash function that is generated and sent from a server to a client to identify the current interaction session.

If Eve can obtain the session ID, she can also, in theory, perform actions, pretending to be the victim. This is known as **session hijacking**. As the session ID needs to be submitted for every POST and GET, it can be easy to obtain it by sniffing or tricking the victim. Hence, there should also be other security measures in place to make the use of session tokens safe.

→ **Smashing the stack.**
Making an overflow such that excess bytes are considered as code rather than arguments.

→ **Software Security Touchpoints.**

→ **SQL-injection.**
If you can't define this, please go get some sleep.

→ **Target of Evaluation.**
ToE. The product under evaluation.

→ **Trojan.**
A piece of software that looks cool, but in reality it causes harm when executed.

→ **Two-channel authentication.**
I fail to see the big differnece from Two-factor authentication but accoding to the book, this is "sending an access code the user via a separate channel"

→ **Two-factor authentication.**
To authenticate, you need "something you have, and something you remember". E.g. a password and a password calculator.

Most companies are sceptical of this. While it does seem to improve security per today's date, it is still prone to real-time mitm attacks.

→ **validation of input.**
Given input ι, filter any bad input b and return ι_{clean} There are multiple types of input validation:

Blacklist-validation do not regard context and trim away any bad characters from input. I.e. one can define a set of bad characters in a set $S = \{', ", \}$, etc. Given input ι, return ι_{clean}, s.t. $\iota_{clean} \cap S = \{\emptyset\}$ The problem with black-validation is that is often easily bypassed, since the attacker can often distort his input in a way that bypasses the filters. Note that the filter only checks for characters, not context.

Whitelist-validation In many contexts, the character ' is considered malicious, however, it is also required in some names, etc. White-validation looks at what structure input should have, and validates accordingly. It is stronger than blacklist-validation in that, for example for dates it is possible to know what structure the input should have , and thereby you can trim from character length and structure.

It is also worhtwile to talk of this as contextual encoding.

Another form of validation is the translation of characters to another typeset. This is typically known as **escaping** input. Before using any input one can e.g. do html-escaping. I assume the reader knows what this is. Character escaping has a recommended order: first do HTML-escaping, then JS-escaping. Finally, it is worth mentioning about escaping that it does not prevent XSS, it just makes it harder to render data as code$^{[why?]}$.

→ **Valet attack.**
Assume that you use a random number in your key to unlock something, like a car. This means that whenever you want to unlock the car, there are different codes, all valid.

If Eve gets to unlock Alice's car every day, Eve can record all keys used to unlock Alice's car. Eventually, when Alice's car run out of memory, it will not remember that the first key has ever been used. Eve then tries to use this first key that she recorded. This unlocks the car.

→ **Virus.**

→.
XSS upload a script that to a server that will be executed by other users. XSS is a popular type of attack and hence there's a lot of terminology...

Stored XSS the malicious script sent by an attacker is stored permanently on the webserver.

Reflected XSS the malicious script sent by an attacker causes an immediate malformed response from the server, but the script will not be stored on the server.

DOM-based XSS typically sends a URL to a user such that his/her site, i.e. DOM, is modified. E.g. if the values for a selection field is specified from the url, modifying the parameters to be scripts instead means that the DOM is modified via XSS.

> *Reflected and Stored XSS are server side execution issues while DOM based XSS is a client (browser) side execution issue*
>
> — OWASP, *https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet*

Mitigations: see validation of input

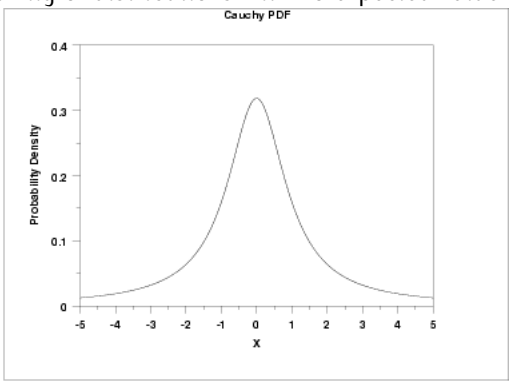
1.8 Statistics

→ **Bernoulli distribution.**
A distribution of parameters, where $\forall x, 0 \leq x \leq 1$, Typically dual, e.g. in coin toss there is heads and tails. If $\Pr_{heads} = k$, then $\Pr_{tails} = 1 - \Pr_{heads}$

→ **Bigvee.**
$\bigvee_{a \in I} P_a$ says that at least one P_a is true. It can also be used for the maximum value in a set.

→ **Central moment.**
The distribution of variable [values] around the mean. This way, you can now if the distribution is spread out or not.

→ **Cauchy distribution.**
A family of distributions with no expected value. They usually look like:

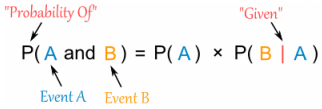


→ **Chernoff bound.**
In a nutshell, it determines a bound on how many times we must perform a trial to know that our random variables represent a majority. E.g. if we trying to determine that a coin is biased (heads/tails), a chernoff bound will say how many times we must flip the coin to know that we have unraveled a bias. In this case, for example, simply flipping it twice will not suffice.

→ **Conditional expectations.**

$$E(X|Y=y) = \sum_{x \in X} x \; P(X=x|Y=y) = \sum_{x \in X} x \frac{P(X=x, Y=y)}{P(Y=y)}$$

→ **Conditional probability.**



→ **Covariance.**
A measure of how much two random variables change together.

→ **Expected value.**
$E[X] = \sum_{s \in S}^{\cdots} X(s) \cdot \Pr(\{s\})$
$E[X] = \sum_{s \in S}^{\cdots} X(s) \cdot \Pr(X = x)$

→ **hyperplane.**
A plane (surface) that has one less dimension than it's ambient space, i.e. the space around it. E.g. a hyperplane for 3-d dims is only defined 2D.

A hyperplane will therefore act as a separator. Imagine a holding a square in the middle of a ball.

→ **Gold standard.**
In medicine and statistics, gold standard test refers to a diagnostic test or benchmark that is the best available under reasonable conditions. It does not have to be necessarily the best possible test for the condition in absolute terms. For example, in medicine, dealing with conditions that require an autopsy to have a perfect diagnosis, the gold standard test is less accurate than the autopsy.

→ **Ground truth.**
to the accuracy of the training set's classification for supervised learning techniques. This is used in statistical models to prove or disprove research hypotheses. The term "ground truthing" refers to the process of gathering the proper objective data for this test.

→ **Normal (Gaussian) distribution.**
A distribution that is centered around a value. E.g. when you collect people's height, there will be many around the average, and fewer and fewer to the sides.

→ **Indicator variable.**
Indicator variable: 0 or 1 for whether an element is selected or not.

→ **Linearity of expectation.**
$E[X] + E[Y] = E[X + Y]$
$[\sum_{x \in S}^{\cdots} X(s) \cdot \Pr(X = x) + \sum_{y \in S}^{\cdots} X(s) \cdot \Pr(Y = y)]$
$= [\sum_{s \in S}^{\cdots} a \cdot \Pr(Y = a) + a \cdot \Pr(X = a)]$

Theorem 1 (Likelyhood that both X and Y occur in S)**.**
$E[X] \cdot E[Y] = E[X \cdot Y]$

Proof. From Expected value:
$E[X \cdot Y] =$
$\sum_{z \in S}^{\cdots} z \cdot \Pr(X = z \text{ and } Y = z) =$
$\sum_{x \in S}^{\cdots} \sum_{y \in S}^{\cdots} x \cdot y \cdot \Pr(X = x \text{ and } Y = y) =$
$[\sum_{x \in S}^{\cdots} x \cdot \Pr(X = x)] \cdot [\sum_{y \in S}^{\cdots} y \cdot \Pr(Y = y)] =$
$E[X] \cdot E[Y]$ □

→ **Markov Property.**
Memoryless stochastic process - what happened at point of time X does not matter for times $Y \geq X$.

→ **Poisson distribution.**
expresses the probability of a given number of events occurring in a fixed interval of time and/or space if these events occur with a known average rate and independently of the time since the last even.

→ **Probability Mass function.**
A function that gives a probability that a variable is exactly equal to some value.

→ **Supervised learning.**
the machine learning task of inferring a function from labeled training data. The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal).

1.9 Operation Research

→ **Corner point feasible solution(CPF.**
A solution that lies at the corner of a feasible region (solution space) If a solution has only one feasible solution, it must be a CPF

→ **KKT-conditions.** 1. $\frac{\partial f}{\partial x_j} - \sum_{i=1}^{m} u_i \frac{\partial g_i}{\partial x_j} \leq 0$

2. $x_j^*(\frac{\partial f}{\partial x_j} - \sum_{i=1}^{m} u_i \frac{\partial g_i}{\partial x_j}) = 0$
3. $g_i(x^*) - b_i \leq 0$
4. $u_i(g_i(x^*) - b_i) \leq 0$
5. $x_j^* \geq 0$
6. $u_j^* \geq 0$

→ **Feasile region.**
Aka solution space. The set of all valid solutions, i.e. those that do not violate constraints.

→ **Slope-intercept form.**
Instead of expressing an objective function, show it as an expression of your parameters. (e.g. $x_1, c_{i,j}$)

1.10 C++

const <C>& function { return FOO; } means it will return a constant reference to C
<C>& function const { return FOO; } means it will not modify any data in C.
const char *var is a pointer to a constant value, not a const pointer (char const *var).
std::vector<T>will typically allocate space in 2^n, where n is minimal. Whenever n increases, a new buffer is allocated, which means that it will have to iterate over all it's former elements, giving $2^n + 1$ iterations. Hence, one should pre-allocate the space needed when possible.

Whenever you use cout to output something, ostream buffers it but does not send it to the output device immediately. Only when the buffer is flushed will the output get sent to the destination. This can cause trouble if e.g. twidth he buffer is full (usually 512 bytes).

→ **Rule of three.**
If you implement at least one of the following, you should implement the others, too:

- Destructor
- Copy constructor
- Copy assignment operator

1.11 Haskell

→ **=>.**
Every expression preceeding the =>is a **class constraint:** it forces types and memberships

→ **->.**
Used to separate variables and return types in functions.

→ **Guards.**
Guards are indicated by pipes that follow a function's name and its parameters. They evaluate to true or false, and then the following function body is used in case of true.

→ **return.**
Turn a type into IO. E.g.

```
getFilename :: String -> IO String
getFilename file = do
    bool <- doesFileExist file
    if bool
    then return file
    else return "batman.wav"
```

Where the safe "file" is returnned as IO String

1.12 More words and def...

→ **Concurrency.**
To execute several unrelated tasks at the same time. E.g. on a game-server one thread deals with chatting, one deals with connections, etc. One important aspect of concurrent threads is that they are Nondeterministic programming. That means that you can predict their state pre-emptively. That is also why we usually do so much thread joining etc. See also Parallelism.

→ **Currying.**
the technique of translating the evaluation of a function that takes multiple arguments (or a tuple of arguments) into evaluating a sequence of functions, each with a single argument (partial application)

E.g. $f(x,y)--->h(x)=y->f(x,y)$. Here, $h(x)$ is a curried version of f, and the $->$ is a function that maps the result from h to f.

→ **Folding.**
Take a list, apply a function and reduce the list to a single value. (this is the python's reduce)

→ **High order functions.**
can take functions as parameters and return functions as return values.

→ **Include guards.**
Avoid re-invoking a header multiple times: enforce that they are only loaded once. In C++ (11) you can use the

```
#pragma once
```

→ **Nondeterministic programming.**
A nondeterministic programming language is a language which can specify, at certain points in the program (called "choice points"), various alternatives for program flow. Unlike an if-then statement, the method of choice between these alternatives is not directly specified by the programmer; the program must decide at run time between the alternatives, via some general method applied to all choice points

→ **Lock order inversion.**
Entity A ackquires a lock on X, entity B ackquires a lock on Y. Now, X depends on something in Y and verca visa. The programs will now deadlock - each waiting for the other to finish their task.

→ **Parallelism.**
To use multiple threads to solve one problem. See also Concurrency.

→ **Strict variable.**
A variable with a determined typed. E.g. writing "int myvar = 5" instead of just "myvar = 5" (non-strict).

Chapter 2

Algorithms and Methods

2.1 Big Data

Data is aquired by sensor, simulations or feeds. Can we filter or compress the feed? Can we trust the sensors (are they faulty?) Information extraction: seeing structure in data, amending it where necesessary. Aggregation: heterogenous sources needs to integrate. Modeling. Note that the data can be noisy. Still might be worthy to visualize and interpret.

Challenges: heterogenity, inconsistency, incompleteness, scale, timeliness... Privacy is also becoming an increasingly important concern now that we get data from just about everything.

A trend in NoSQL and BDMS is horizontal scalibility over vertica. That is, we prefer having many servers over one powerful one. Across these shards of data we do horizontal partitioning: one file is split across nodes.

Since NoSQL is inherently less structured, its queries are simpler, often a minima to provide CRUD operations.

There are four types of databanks for NoSQL DB's:

1. **Document-based:** store a document and an ID, query by the ID.
   Example: MongoDB.
2. **key-value:** allow custom keys and custom values (e.g. one document can have multiple indexes for multiple values).
   Example: Cassandra, Voldemort.
3. **column-based:** instead of just having documents, you structure each record with columns.
   Example: HBase
4. **Graph-based:** cool stuff. One cool advantage is the simple queries for complex operations.
   Example: Neo4j

→ **Log-structured Merge Tree.**
(LSM) A data-structure optimized for indexed access with a high insert volume. It typically stores a lot of data in key-value form in-memory, and flushes out when exceeding a memory threshold.

→ **Consistent hashing.**
A hashing scheme that scales well. When you add keys (expand domain), you don't need to remap the majority of the keys as you would have to in normal hashing. This can be thought of as extending a simple hashing algorithm $h(x) = mod k$. When we expand the domain of the function, it will simply result in a lower hashvalues that can be used directly without changing other indices.

→ **ACID. Atomicity:** An action will either completely fail or completely succeed, nothing in between.

Consistency: An action will either completely fail or completely succeed, nothing in between.

Isolation: Executing multiple actions simultaneously yields the same result as if all the actions were executed serially (that is, no messing up because of pararell execution).

Durability: If an action is committed, it will remain in the system even if there are power losses, crashes or errors.

→ **Uber task.**
A term used in MapReduce to denote a job so small that pararellizing it will not cause a benefit, thus leaving the job to be resolved in the same JVM as the master.

→ **BASE.**
An alternative way of analyzing DBMS. It is less strict than ACID.

Basic Availibility database should work most of the time

Soft-state replicas don't always have to be consistent

Eventual consistency An update doesn't have to be seen by all peers right away

→ **CAP theorem.**
Working with Big Data involves dealing with inputs so large that conventional methods do not work. This also involves that we have to compromise, not opting for "optimal" solutions like RDMS. We have

Consistency: Whether or not copies of data are the same across all nodes. For example, having a server in London with data X and another server in USA with data X', where X' is intended to be equal to X, but it is not.

Availibility: Whether or not we can guarantee success or failure (alternatively: every request returns a non-error response).

Partition tolerance : If a node goes down, will the system continue to operate?

The CAP theorem states that any big-data system can only acheive two out of tree letters (CA, CP or AP).

Proof. Assume a system has two nodes: A and B. A and B cannot communicate. We write data X to node A and B. Then, we write X' to node A. Following that we want to read X from node B. If node A and B do not talk together, we will not achieve consistency (node B doesn't know X is updated in A). This also means we struggle with availibility: X' has not been written to both nodes. If we do let node A and B talk together, then node B depends on A, so we are not parition tolerant. Hence, achieving all three letters is not possible in this case. □

→ **Five V's of Big Data. Volume:** How much data do we have?

Veracity: Can we trust the data we have?

Variety: What types of data do we read?

Value: Is it really worth while to do big data solutions?

Velocity: How fast is data coming in?

→ **At-least-once ingestion.**
If any message is lost, retransmit it. Thus, we can guarantee that a message will be received. There is a possibility of duplicates.

→ **At-most-once ingestion.**
Every message will be sent, but at most one time. There is a possibility of duplicates.

→ **Exactly once ingestion.**
Every message is delivered once. No duplicates. Hard to implement (you need to retain data for a longer amount of time and have synchronization protocols).

→ **Equi-depth histogram.**
Take query:

```
SELECT * FROM Person WHERE AGE > 24
```

. Here, we can optimize the query by using an equidepth histogram: instead of preparing to return each row, we first process the AGE column and count how many occurences we get. Then, we estimate parallelism, return size, etc, and return queries. If AGE is indexed, we could make estimates like this in $O(1)$.

→ **Equi-width histogram.**
Similar to Equi-depth histogram, except we fix bucket sizes. For example, for an age query we don't return the number of people above a threshold age, instead we bucketize, e.g. everyone under 24 in one bucket, everyone over 24 in another, or splitting per 5 years, etc.

Q: how can a join be affected by whether or not we sort the output?
A: When we do JOINS, we lookup keys from table X and Y. We put together keys from e.g. Y into table X. Therefore, table X needs to be either sorted or use some form of hashing scheme so that we can efficiently find the key in X that matches the key in Y.

If data is assumed ordered, we save a great deal of time. All we need to do is iterate over table Y, and allocate each key to its corresponding bucket from table X. This will be joined data. We will not shuffle table X around much.

Q: discuss joins in relation with parallelism?
A: If we have multiple datanodes availible, we can do mapping parallelized. There is more overhead since we need to merge partitions afterwards, but for large inputs the tradeoff is still positive.

Q: describe diffs in traditional DBMS and data streams.
A: There are some obvious differences in terms of technology. Streaming technology has key uses in fields where automated decisions are necessary. It can be used for controlling prices, shutting down faulting hardware or giving an early warning to monitoring analysts. They will not directly support saving any form of state. Doing so introduces a significant overhead, and it may be worth-while to consider writing to disk first. An interesting side-note is that some RDBMS like PostgreSQL offer At-most-once ingestion analysis, quick processing and horizontal scaling. For applications that need both streaming and storage, this can be a good alternative.

Q: Should file partitions always have the same size? A: It is interesting to note that if a partition is sufficiently small, it can fit into memory. Since memory can dynamic, varying partition size can be desirable to fit as much in memory as possible. Otherwise, it is usually preferable to choose a fixed partition size so that sharding and other operations are easier, and to prevent stalls induced by having some nodes finish up before others due to a smaller partition sizes.
A: ah.

Q: Explain a sliding window in streaming systems
A: Iterating over windows at a time. Entries are iterated mutiple times since they will occur in multiple slides. Old data falls outside of the window because it is no longer relevant.

Google File System One of the first file systems to scale with big data. The main idea is to replicate data exensively to provide high availibility and reduce transmission costs from having to send data on-demand. Data is most often "written once, updated seldom" which allows writes to be inefficient (unlike some other RDBMS). Aso, whenever you modify an entry, you typically prefer appending modifications rather than re-writing it.

Another interesting feature of GFS is that it typically runs on commodity hardware rather than specialized computers. This induces a higher risk of hardware failure, which needs to be dealt with. One exapmle solution is to implement frequent "heartbeats" from commodity computers to a master node.

GFS is optimized for MapReduce; the paritioning scheme makes map-reduce jobs simple.

For security reasons, GFS also obfuscates its data with randomization algorithms.

HDFS Very similar to Google File System. It was intended to be open-source alternative to GFS. It can now support more nodes and has different security/permissions model, often from POSIX.

Consumed by HBase?

Voldemort

> *It is basically just a big, distributed, persistent, fault-tolerant hash table*
>
> *Internet*

Used for LinkedIn and their data. It provides a data-store with key-value pairs. This is simple to deal with, but eliminates possibilities for comlex queries, foreign keys, triggers, etc. Values can be either JSON or whatever format the user desires. The hashmaps can also be in-memory, meaning they're fast.

Alike many otherBDMS, Voldemort focuses more on AP than consistency (see CAP theorem). The level of consistency can, however, be adjusted. Often, however, it is left to the client, who receives multiple copies of data if there is a risk of concurrent writes having corrupted a value. What is interesting is that they do not satisfy all ACID properties; data isolation is not ensured.

Voldemort is different than e.g. HBase in that they focus on having a large amount of writes in their systems. In Hbase this is not prioritized, henceforth it is slow. Also, with the use of hashmaps, their data is inherently more unstructured.

Voldemort has low latency, but not the best throughput (like Cassandra).

MongoDB

*MongoDB (from hu**mongo**us)*

Wikipedia

MongoDB stores data in BSON, i.e. binary JSON. It can index its data in multiple ways:

Array Each entry of an array can be indexed

Compound Indexes built on multiple fields such as "Name + Age"

Geospation Coordinate points that refer to a location, etc

Partial Indexes that reflect data. For example get all customers that did something the last 24 hours

Sparse Index only documents that has a certain field (different from partial where the value of the field is used)

TTL Indices that expire after a given amount of time

Text Search Index a text and provide a key that indicates the text's relevance in accordance with query

Unique Normal ones!

Naturally, given the amount of indexes that can be used, there are also many queries. Some examples are geospatial queries, range queries and MapReduce. The latter is often executed as a javascript code query, a nifty feature of MongoDB.

One other nifty feature of MongoDB is autosharding. As long as you provide clusters and computers, MongoDB is capable of balancing the load for you.

A downside of MongoDB (andBDMS in general) is that joins aren't supported natively.

With replicas, MongoDB always selectes a primary as the main provider for data.

MongoDB compresses its data.

MongoDB is ACID compliant at a document level. However, for updating all of its replicas, there is a possibility (albeit small) that an error will occur. A use case that is particulary relevant is customer purchase: first remove item from the inventory, then add that item to a customer. Now you need to do two actions in one go. Normal RDBMS will handle this with transactions, however, in MongoDB there is a chance that you are able to remove the item from inventory, but not add it to the customer. If this happens, you are not ACID at a higher level, although you have ACID at document level. The property failing is (like Vodlemort) isolation. In regards to CAP theorem, MongoDB offers eventual consistency

Cassandra Developed at Facebook for inbox search. High throughput, but at the price of high write and read latency. Nodes are independent, but connected to all other nodes.

Key-value ish, however, the value is highly structured (unlike other big data management systems). Very scalable, as mostBDMS are. Uses Consistent hashing to do so, and has different modes for replication such as "rack aware" and "datacenter aware" schemes.

Neo4j Intuitively cool, Neo4j is a graph database. It is ACID compliant (notably, unlike the systems discussed above it has support for transactions). Queries are simple, even for complex joins. Unlike EER diagrams (which Neo4j might be similar to), vertices may have no type and relationships are directed.

AsterixDB Like all other BDMS, AsterixDB comes with its own superflous words to describe normal things. The top-level node is called a *dataverse*. Other papers describe dataverses as databases. Unlike other BDMS, AsterixDB has a strict type policy, which means that it will do type-checking on that data inserted into a dataverse. It is possible, however, to make a loose definition of type, thus allowing random data to enter the dataverse (naturally called having "wiggle room").

Instead of using JSON, AsterixDB uses something called ADM: an extension of JSON. Keep in mind that it is still possible to have strict requirements for the structure of JSON data.

All data-records have a key. Keys can be composite or optimized for fuzzy string search. Each dataset has a B^+-tree with key-value pairs.

AsterixDB can read non-indexed data, e.g. loading a CSV-file. This will be mounted read-only. This helps them read data-feeds just as well as data from e.g. HDFS. However, AsterixDB does not operate directly on the stream directly, it requires structuring data before queries are executed. The neat benefit from this is that AsterixDB hides complexity; quering a feed is exactly the same as quering a stored dataset from a developer point-of-view.

Continuing in the BDMS-framework fashion, AsterixDB defines its own query language. AQL is similar to XQuery (for XML), except they claim AQL is simpler. AQL is compiled down to algebricks which can be optimized. The output is a (often parallel) Hyracks job.

Keys are stored as 2-way Log-structured Merge Tree, which opens up for rapid insertions (aka dealing with high ingestion).

AsterixDB supports transactions and is thus ACID-compliant.

Storm Developed by Twitter, storm is real-time and fault-tolerant API/framework.

To analyze storm we talk about data as directed edges in a graph, moving between vertices that represent computations. Data is in the form of a stream of tuples. Each graph is for whatever reason called a topology. In each vertex there are two disjoint sets: a spout and a bolt. The spout receives the data, and the bolts process them. Each bolt may direct its data back to a spout if it wants to (achieving cyclic iterations).

Storm uses the concept of master and worker-nodes. The master node is responsible for delegation. A worker node can have multiple jobs, but only one topology at a time. Worker nodes have monitors called supervisors that talk to the master node. Every relationship has synchronizations: the worker sends heartbeats to its supervisor, the supervisor synchronizes its master, and the supervisor synchronizes its topology.

Storm has a summingbird that can optimize queries for you.

Storm has At-least-once ingestion and At-most-once ingestion. The former is ensured by adding "acker" bolts to tuples outputted from a spout. This is done by appending an ackerbolt to the topology and a 64-bit field-value to each tuple. The latter is ensured by default if At-least-once ingestion is disabled.

Yet Another Resource Negotiator YARN has a resource manager per cluster. Inside each each cluster there are multiple node managers. The node managers manage containers, which is the equivalent of a process.

Clients can make resource requests, asking for locality of nodes or a given amount of memory. This is stated up front, but can also be changed dynamically.

Allocation can be per-user, per-session or interchanged between users. Per-user is often used in MapReduce, where the client is given a set of nodes that are used once, and then released. A per-session is used for iterative jobs (common in e.g. Spark), so that after finishing the first stage of a job, the same nodes are allocated for the second iteration as well. The third is more appropriate if you for example have many jobs from different companies or applications and want to fix or avoid allocation errors (giving each company a set of nodes, for example). Also common for long-term jobs that go for several days and/or months.

Comparing YARN to MapReduce 1, we see a bigger separation between entities. There is no single jobtracker: its job is separated into resource manager, application masters and timeline servers. The improved structure allows them to scale more, provide higher availibility (eliminating/reducing single points of failure), utilize resources more effectively, and do multitenancy. The latter means that you can have multiple jobs on the same clusters. Furhtermore, it is no longer just MapReduce jobs that are executed, but other frameworks like Spark and Storm enter the scene.

YARN has three schedulers. There is FIFO, Capacity and Fair schedulers. FIFO is simple to configure but doesn't scale well with large clusters. Capacity retains simplicity: it provides each user with a small, reserverd amount of resources. This pool of resources is seen as a small and private FIFO queue. If availible, the job will be upgraded with more resources to do its computations. Capacity is a good scheduling algorithm for multi-company uses of a WSC datacenter. Fair scheduler dynamically allocates resources to jobs. When a single job is on the cluster, it receives full resource utilization. When another job comes in, it receives half of the resources, and so on.

Spark Arguably better for analytics than queries. This is because it primarily works on HDFS.

One of Spark's genius concepts is to create small subsets of data seen as read-only objects with lazy evaluation. This means we can track *lineage* and restore data upon failure. Also, their *resilient distributed datasets* (RDD) are great for pararell and iterative executions of data. This is seen for example by *persisting* RDDs, saving RDD-state and re-using them in several computations. Persistance can be done in-memory, to disk or both. Persisted RDD are immutable to ensure consistency. This is not a feature seen in e.g. distributed shared memory (DSM). The advantage for the latter, however, is that modifying data is easier.

PageRank Initially, give each node a rank. For example all nodes start with rank $\frac{1}{N}$, where N is the number of pages. Now, iterate over all the nodes, giving each node x rank in iteration t:

$$PR_t(x) = \frac{1-\lambda}{N} + \lambda \sum_{y \to x} \frac{PR_t(y)}{out(y)} \tag{2.1}$$

The LHS of PageRank is the probability of being at this given node. The RHS is the probability of being at any given node, multiplied by its pagerank *for the current iteration*, for all connecting vertices. The *out* for each node is constant, and refers to how many edges there are pointing out from node y. The intuition is that a node may have a high pagerank (which makes x more likely), however, if y has many nodes, it reduces the chances for x (which is why we divide PR over out).

Note that pageranks should, in the end, sum up to 1 (or 100%).

2.1.1 MapReduce 2.0

Client asks resource-manager for resources. Resource-manager verifies everything is OK, and sends back info about an availible container. Client is happy, and uploads JAR. If the first container (appmaster) realizes this is not an ??, it will ask resource-manager for more buddies. It will prefer local nodes, but the resource-manager can overrule priorities if there are conflicts. Resource-mananger complies. Now the job is running. When finished, application managers report that we are done, and releases its resources. As an extra side-note, it might be worth noting that the reduce-part can begin before all maps are finished (to save time). Failure results in re-trying the code on a different container n times before giving up. Failure can either be tolerated up to x percent or result in halted execution. AppMaster can fail and be restarted from its log files. This is done by the resource-manager. Since the resource-manager plays such a critical role, its function is often duplicated across several nodes to prevent having a single point-of-failure. All of the information processed by resource-managers are also stored in high-availibility systems like HDFS or using ZooKeeper.

Shuffle When a Map has been performed, we need to output data to reducers. Before this happens, we need to *shuffle*. This is sorting the data by which reducer it will end up with. This is done by having a small, circular buffer being filled up. When it reaches a given size it is spilled to disk. The spill files are re-iterated in the end to create a bigger end file. When the *shuffle* is finished the *copy phase* begins: take data from mapper to reducer. Upon arriving by the reducer, the data is sorted again.

A nice thing to keep in mind is to leave the shuffle phase to have as much memory as possible. This will typically give the best results for MapReduce jobs.

In mapreduce 1.0, map and reduce part always had the same resources (not dynamic). Also, you could only execute mapreduce jobs (not complex shuffles, etc).

Joins A Join is a means for combining fields from two tables by using values common to each. If one table is small, we often like to distribute it to all nodes such that we can quickly lookup all matching entries. If not, we need to be clever.

→ **Reduce-side join.**
On the map-side of things, we simply sort the data by join-key. E.g. a mapper A will give reducer A' all entries with key K. Then, reducer A' performs another sort, joining entries from two or more tables on K.

→ **Map-side join.**
Similar to Reduce-side join, but we avoid the second pass of sorting. A few conditions are necessary to do the join in the first stage of our job.

1. Data is already sorted by join key inside partitions
2. Input datasets have same number of partitions (difficult if doing joins for > 2 tables.

TODO: tougher questions: given condition X and Y, which framework would you use? when use streaming vs non-streaming vs hybrid? TODO: a join in mapreduce TODO: transitioning from RDBMS to BDMS TODO: review lectures

2.2 Algorithms

2.2.1 Cristofeledes algorithm

Objective: solve TSP with $\frac{3}{2}$-approximation

1. Construct an MST
2. Extract the odd degree vertices
3. Find a matching, using odd degree vertex from (2)
4. Combine the edges from the tree with those in the matching
5. Form a eularian circuit
6. Short the the traversal path to form a hamiltonian cycle

2.2.2 Double-tree algorithm

2-approximation

1. Construct MST
2. Give each node a redundant entry (graph can now be traversed like an Eularian graph)
3. Traverse the nodes from start to finish (DFS), but only retain the first time occurence of a city

2.2.3 Dijkstra

Assign all nodes distance = ∞ Start at given node s. Assign all neighboring nodes their distance from s to n. Proceed to the lowest cost. Repeat. Whenever a better match is found, use the new path instead. The result is a path from s to t, or an MST.

2.2.4 K-center

2-approximation. A form of clustering.

1. Pick arbitrary $i \in V$
2. $S \rightarrow \{i\}$
3. pick other node furthest away from k, add to S
4. repeat, furthest away from all previous picked nodes.

2.2.5 Kruskal's algorithm

Always choose cheapest edge that does not include two pre-discovered nodes. Returns MST.

2.2.6 Nearest addition

2-approximation

1. Find two closest cities
2. Traverse from i to j and back
3. Repeat, consider from each node

2.2.7 Prim's algorithm

Start at a given node s. Choose the cheapest edge. Now repeat, just that you consider s and the node you added. Etc. Never choose an edge that leads to a pre-discovered vertex.

2.2.8 List scheduling algorithm

Whenever a machine is idle, assign it a job. This algorithm runs $\leq 2 \times OPT$.

Proof. Note that $OPT = \frac{\sum p_i}{m}$ We know that the last job, l, starts at time t. Since we've always assigned jobs whenever we could, that means that all machines have so far been busy. Hence we know:

$$t \leq \frac{\sum p_j - p_l}{m} \leq OPT - \frac{p_l}{m}$$

Since OPT is lower bounded as the average work done by each machine.

We can now bound the maximal finishing time:

$$C_{\max} \leq t + p_l \leq OPT + p_l \times (1 - \frac{1}{m}) \leq (2 - \frac{1}{m}) \times OPT$$

□

2.2.9 Longest processing time rule

As List scheduling algorithm, just that you first sort the jobs in order of length; put the longest jobs first. This algorithm has approximation ratio $\frac{4}{3} \times OPT$

2.2.10 Shortest remaining time, SRFT

In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. This algorithm is applied to Pre-empty schedule schedules

2.2.11 Knapsack DP

Fill out two arrays of $n \times B$, where B is the capacity of the knapsack. In row i, consider item b_i against the capacity given in column j. If $j \geq w_i$, compare the value of $A_{i,j}$ to $A_{i-1,j}$ and use whichever is greater. For all cases where $b_i \geq j$, consider which is better: to use $b_i + A_{i-1,j-w_i}$, or retain the value above. The last equation is: to use current item + whatever we can fit in on the remaining weight, given by the line above. Make sure to simeoltaneously maintain a "keep-array" that gives 1 or zero, indicating whether or not you've used item b_i.

In backtracking, start in the lower right of the keep array. Whenever you get a 1, include item i and decrement i and j by one. Whenever you get a zero, decrement i alone.

The above mentioned runs in $O(n \times W)$, where W is the capasity of the knapsack. The algorithm is *exact*, but pseudopolynomial, since $W = \sum w_i$, which in binary becomes $\log_2 W$, which hence becomes $O(\log_2 W^n)$. An FPTAS, fully polynomial approximation scheme exists, see book.

2.3 Programming Security

2.3.1 Challenge-Response

You insert a car key into a car engine. The engine would now like to know that this key is valid.

Let K be the key, and E the engine. Then:

$$E \to K : N \tag{2.2}$$
$$K \to E : \{E, N\}_K \tag{2.3}$$

Here, N is a challenge sent by the engine to the key in step (1). The key responds by encrypting the text. Iff the engine can decrypt and get valid number, will it start.

2.3.2 Needham-Schroeder Protocol

A primitive, three-way Key-distribution process. Also known to inspire Kerberos[1], the security protocol in windows. The word *primitive* is used to emphasize that the protocol comes from an age where attacks were mostly thought of as external-to-internal penetrations, i.e. one needs to assume that attacks come from external networks, by people without means of authentication. Hence, the protocol fails to defend against users of it's own system.

Having defined key-distribution in section , the following should be somewhat intuitive:

$$A \to S : A, B, N_A \tag{2.1}$$
$$S \to A : \{N_A, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}} \tag{2.2}$$
$$A \to B : \{K_{AB}, A\}_{K_{BS}} \tag{2.3}$$
$$B \to A : \{N_B\}_{K_{AB}} \tag{2.4}$$
$$A \to B : \{N_B - 1\}_{K_{AB}} \tag{2.5}$$

The difference from plain Key-distribution is that we here introduce the concept of Cryptographic nonces. In step 1, Alices attaches her nonce N_A when talking to Sam. The nonce will be used in Sam's reply. That way, Alice can know that she is not recieving an old/corrupted message.

The entitiy $\kappa = \{K_{AB}, A\}_{K_{BS}}$ represents the the shared key between Alice and Bob, encrypted such that only Bob can read it (denoted by the subscript $_{K_{BS}}$). K_{AB} is the key, and the A is included such that Bob knows it is inteded for his use to Alice. In step 3, Alice sends κ to Bob. Bob can decrypt with his key, given to him earlier by Sam. Finally, to ensure that no data has been corrupted by Eve or other errors, the last two steps verify the legimiticay of the keys. Should Alice fail to return back $N_B - 1$, that is, a nonce from Bob−1, then Bob can know that he is not talking to a valid user.

One easy to understand flaw in this system is that if someone is able to obtain Alice's identifier, they could easily impersonate Alice. The only required step is to contact Sam, get keys, and then communicate. Alice cannot be aware of this since she never initiated the requests. To revoke the damage done, Sam would have to keep logs of every issued key and thereafter alert everyone who has recieved Alice's key that she was compromised.

[1] originally from Greek/Roman mythology, a three-headed dog, or "hellhound", which guards the entrance of Hades.

2.4 Vim

2.4.1 Search and Replace

`v/pattern/d`	delete every line not matching the pattern
`g/pattern/d`	delete every line matching pattern
`s/pattern/subst/flag`	substitute pattern with string, flag to specify how

Vim-regex

`\{-}`	similar to .*, except that it is not greedy. The hyphen can also be of form n,m where n denotes the minimal,maximum count respecitvely.
`Prefix_pattern\zsMid_pattern\zeSuffix_pattern`	where: **Prefix pattern** is a part of the match we do *not* care about. This is delimited by `\zs` (pattern begin). **Midpattern** is what we are replacing. This is because of the `\zs` and `\ze` surrounding. **Suffix pattern** will also not be replaced because of `\ze`. Note that `\ze` and `\zs` can be used individually.

Examples

delete every line not matching format 0.00 0.099, and keep newlines	`%s/^$/AB0\.1/g | v/[0-9]\./d | %s/AB0\.1//g`
delete trailing whitespace	`%s/\s\+$//g`
leading	`%s/^\s\+//g`

2.5 Debugging with GDB

Use **bt** to get a backtrace. Example:

```
#0  0x00007ffff48daf7c in std::string::assign(std::string const&) () from /usr/lib/gcc/x86_64-pc
-linux-gnu/4.7.3/libstdc++.so.6
Python Exception <type 'exceptions.IndexError'> list index out of range:
#1  0x00005555555925a7 in image_psb::ImageSolver::doImageDisplay (this=0x7fffffffcb40, mapFiles= std::map
    sImageRoot="../test_media/") at /home/andesil/PSB/src/./image2.cpp:489
#2  0x0000555555591c99 in image_psb::ImageSolver::renderImages (this=0x7fffffffcb40, sImageRoot=
"../test_media/",
    vIf=std::vector of length 3, capacity 3 = {...}, cImagePath=0x55555563e50a "/image/") at /home/andesi
#3  0x0000555555585b4b in main (argc=5, argv=0x7fffffffd888) at /home/andesil/PSB/src/main.cpp:201
```

Here, each number is called a *frame*. You can inspect each frame by typing "**info frame #n**", where #n is the number of the frame. To get a local inspection (set scope), type "**set frame #n**". Then it makes more sense to type commands like "**info locals**" and "**info args**".

To only show some items in the backtrace, type "**bt -n**", where n is the number of frames, from bottom that you want to include.

Chapter 3

Notes and Thoughts

3.1 Algorithms

The performance guarantee of a primal-dual algorithm provides an upper bound on the Integrality gap of an integer programmin formulation. The Integrality gap also gives a lower bound on the performance guarantee that can be achieved via a standard primal-dual and analysys.

Primal dual is good for when there might be exponential problem sizes in the inout.

General method:

1. Determine decisions
2. Define the value of making those deciosion
3. Decide bounds on OPT
4. Prove that decision bound OPT by α

3.2 Programming Security

It's considered difficult to protect against social engineering. Some of the problem is that you questioning everyone takes time, training everyone takes time, and that since being sceptical takes more time for employees, people arent' going to do it. The best way to protect against it is to never trust anyone, and build autonomous systems.

The problem about centralizing data is that once a breach is made, the effects can be more severe. If a small datasource is compromised occasionally, the negative impacts might still be small enough to be handled. However, it is easier making centralized systems safe against small crimes.

Passwords are good to authenticate, but they're easy to guess and people are not good at ensuring policies for passwords. When you ask for secure passwords, people tend to store them in an unsafe fashion such that the **security at the expense of usability is usability at the expense of security**.

In general, protocols may be defeated byy changing the environment that they operate in. Most protocols make some assumptions about how things work, as soon as this is modified, the protocol might break.

A cool form of attack: if a privileged user has "." in his path, and you could put in a piece of software there, like "ls", you could trick the admin into executing it. Since the "ls" in "." might be preferred, he will then launch a program with elevated privileges, that you might have put there.

Appendix A

Greek alphabet

A.1 Conventional letters

Greek	Latin	spelled greek	typical usage
α	A	alpha	constants
β	B	beta	
γ	Γ	gamma	
δ	D	delta	given functions
ϵ, ε	E	epsilon	
ζ	Z	zeta	
η	H	eta	
θ, ϑ	Θ	theta	
ι	I	iota	
$\kappa, \varkappa$	K	kappa	
λ	Λ	lambda	
μ	M	mu	
ν	N	nu	
ξ	Ξ	xi	
o	O	omicron	bounds
π, ϖ	Π	pi	
ρ, ϱ	P	rho	
σ, ς	Σ	sigma	sum
τ	T	tau	time shift
υ	Υ	upsilon	
ϕ, φ	Φ	phi	wavelet
χ	X	chi	
ψ	Ψ	psi	
ω	Ω	omega	

A.2 Other symbols

∂

$\hbar$
$\imath$
ℓ
$\Re\mathfrak{e}$
$\Im\mathfrak{m}$
$\wp$
∇
$\square$
∞
$\aleph$
$\beth$
$\gimel$