EP24: Environment-friendly languages. Also...

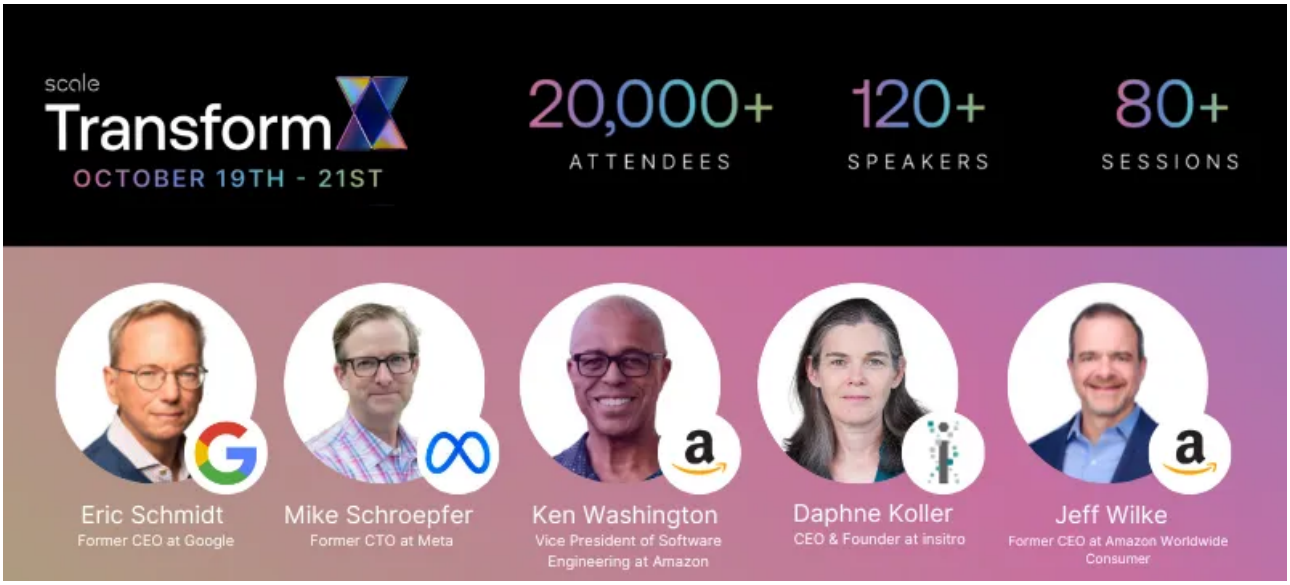
ALEX XU

SEP 17, 2022

In this newsletter, we'll cover the following topics:

- Resiliency patterns
- Environment-friendly languages
- Designing a permission system
- Back-of-the-envelope estimation
- Ways to generate distributed unique ID

The AI event of the year is quickly approaching (sponsored)



We're talking about TransfromX, a free virtual conference where you'll hear from 120+ technology leaders from companies like Google, Meta, OpenAI, DeepMind, Amazon, and many more

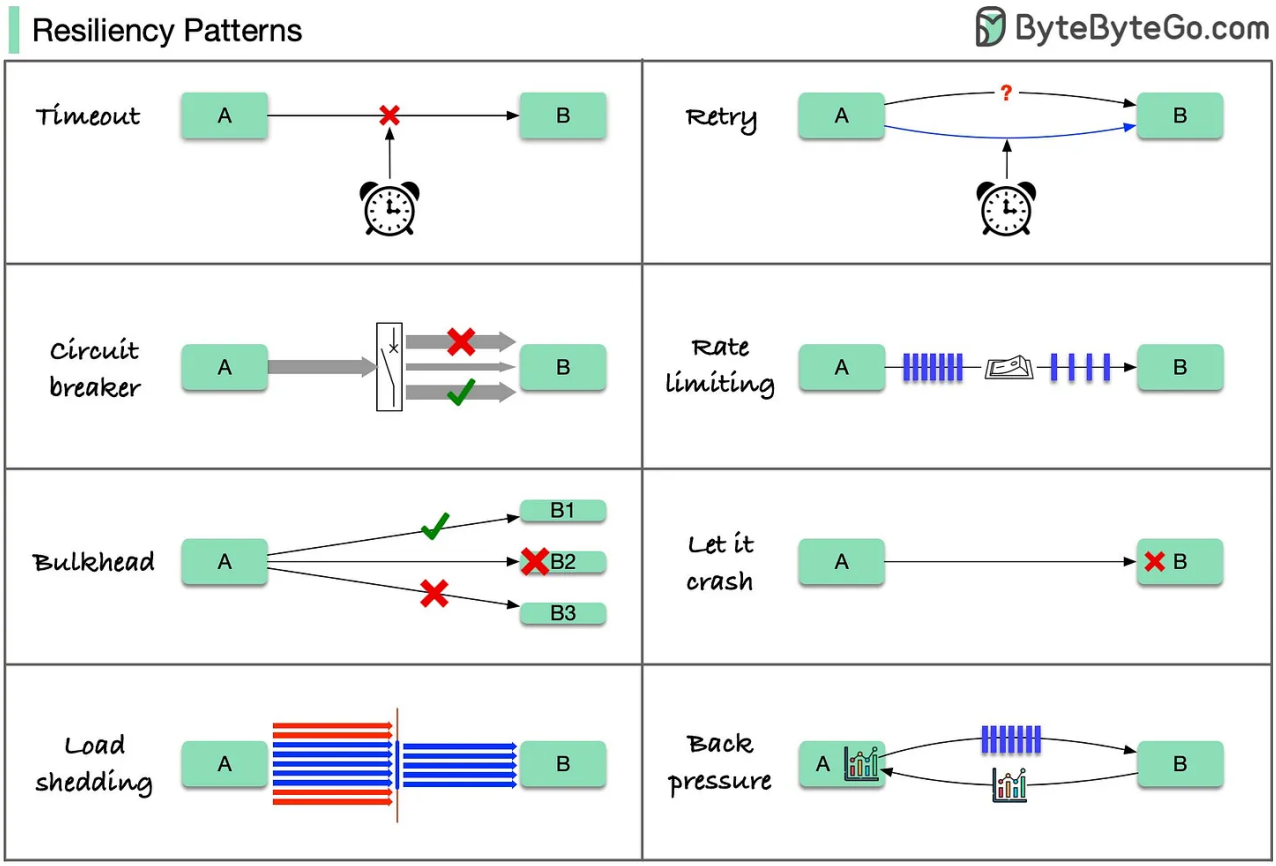
The best part, it's ***FREE***.

On Oct. 19 - 21st, TransformX by Scale AI will explore how AI will power eCommerce, AI applications for healthcare, NFT marketplaces, and more. Explore 80 sessions including fireside chats, hands-on workshops, and keynotes.

We even have a special offer for ByteByteGo subscribers. Be one of the first 50 to register and get a FREE TransformX tee! We'll send you an email once you do.

Resiliency Patterns

Have you noticed that the largest incidents are usually caused by something very small?



A minor error starts the snowball effect that keeps building up. Suddenly, everything is down.

Here are 8 cloud design patterns to reduce the damage done by failures.

- Timeout
- Retry
- Circuit breaker
- Rate limiting
- Load shedding
- Bulkhead
- Back pressure
- Let it crash

These patterns are usually not used alone. To apply them effectively, we need to understand why we need them, how they work, and their limitations.

What Are the Greenest Programming Languages?

The study below runs 10 benchmark problems in 28 languages [1]. It measures the runtime, memory usage, and energy consumption of each language. This take might be controversial.

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

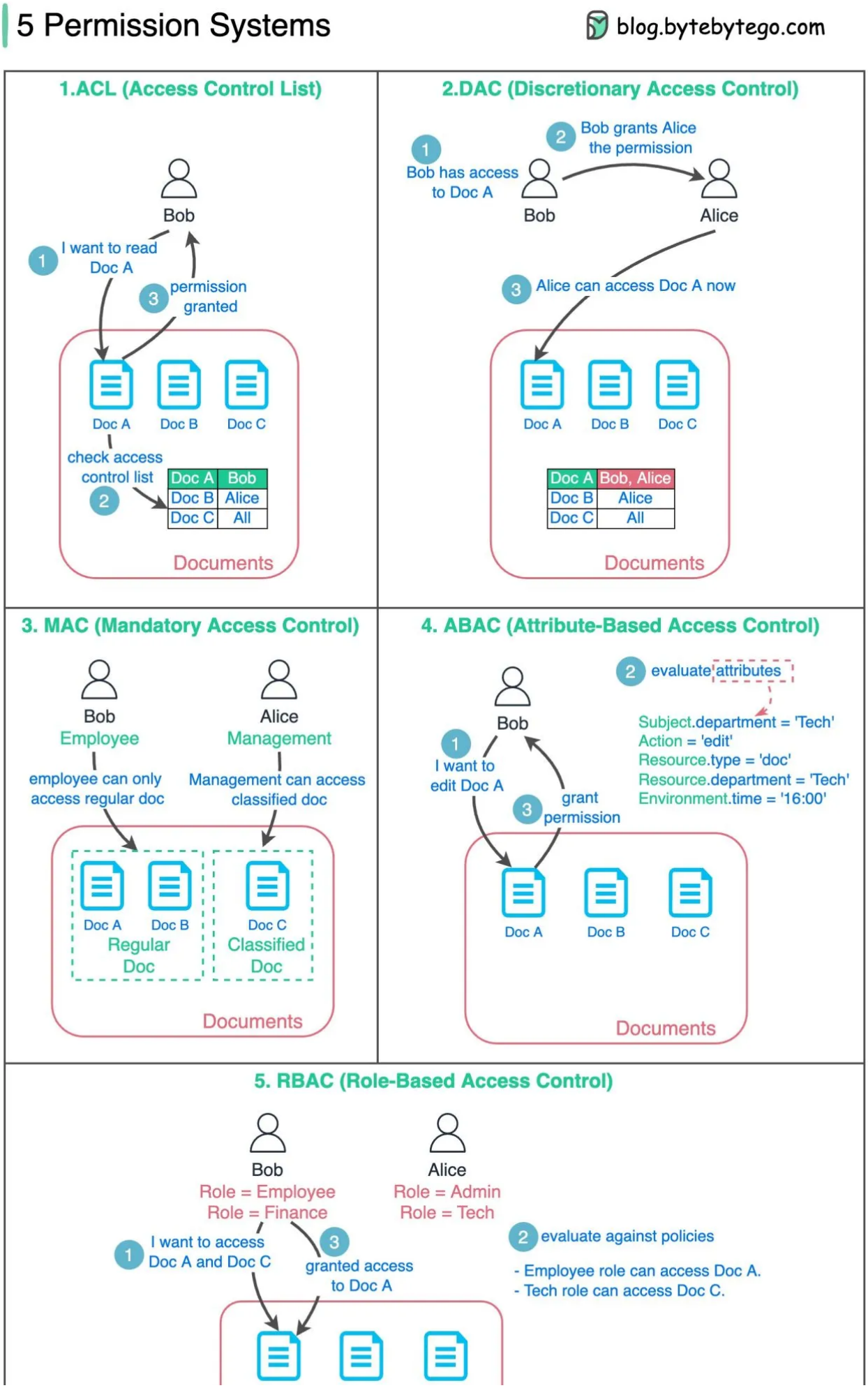
"This paper presents a study of the runtime, memory usage, and energy consumption of twenty-seven well-known software languages. We monitor the performance of such languages using ten different programming problems, expressed in each of the languages. Our results show interesting findings, such as slower/faster languages

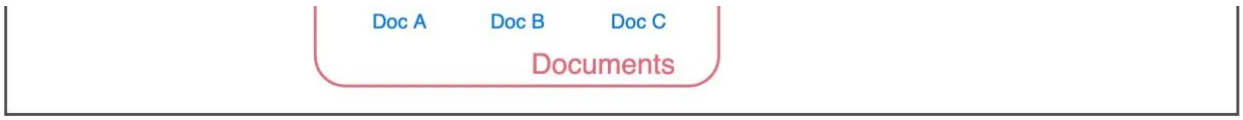
consuming less/more energy, and how memory usage influences energy consumption. We show how to use our results to provide software engineers support to decide which language to use when energy efficiency is a concern". [2]

[1] https://lnkd.in/eYpvP3Dt
[2] https://lnkd.in/eczQYnHD

How do we design a permission system?

The diagram below lists 5 common ways. 👇


5 Permission Systems
blog.bytebytego.com
1.ACL (Access Control List)
Bob
1 I want to read Doc A
3 permission granted
Doc A
Doc B
Doc C
check access control list
2
Doc A | Bob
Doc B | Alice
Doc C | All
Documents
2.DAC (Discretionary Access Control)
1 Bob has access to Doc A
2 Bob grants Alice the permission
Bob
Alice
3 Alice can access Doc A now
Doc A
Doc B
Doc C
Doc A | Bob, Alice
Doc B | Alice
Doc C | All
Documents
3. MAC (Mandatory Access Control)
Bob
Employee
Alice
Management
employee can only access regular doc
Management can access classified doc
Doc A
Doc B
Regular Doc
Doc C
Classified Doc
Documents
4. ABAC (Attribute-Based Access Control)
2 evaluate attributes
Subject.department = 'Tech'
Action = 'edit'
Resource.type = 'doc'
Resource.department = 'Tech'
Environment.time = '16:00'
Bob
1 I want to edit Doc A
3 grant permission
Doc A
Doc B
Doc C
Documents
5. RBAC (Role-Based Access Control)
Bob
Role = Employee
Role = Finance
Alice
Role = Admin
Role = Tech
1 I want to access Doc A and Doc C
3 granted access to Doc A
2 evaluate against policies
- Employee role can access Doc A.
- Tech role can access Doc C.

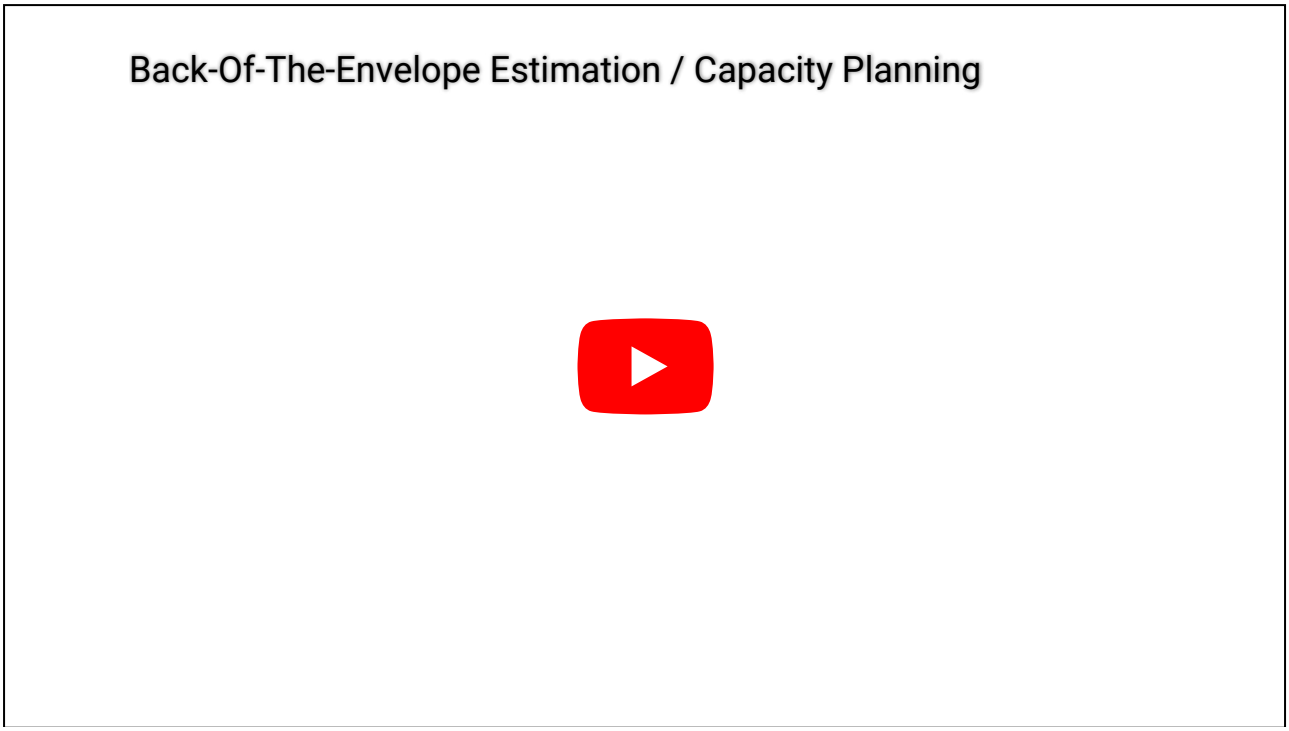


1. ACL (Access Control List)
   ACL is a list of rules that specifies which users are granted or denied access to a particular resource.

   Pros - Easy to understand.
   Cons - error-prone, maintenance cost is high
2. DAC (Discretionary Access Control)
   This is based on ACL. It grants or restricts object access via an access policy determined by an object's owner group.

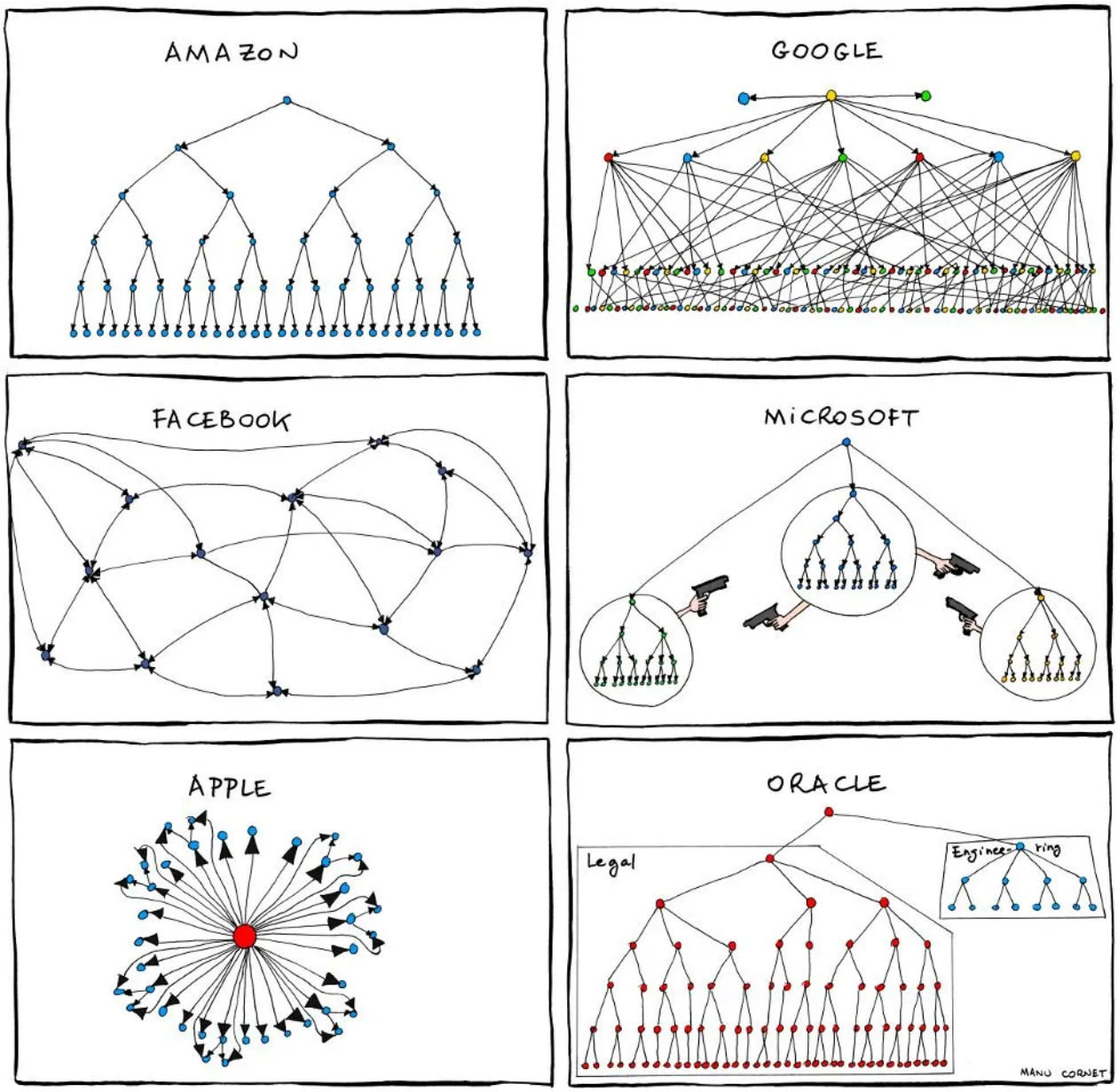
   Pros - Easy and flexible. Linux file system supports DAC.
   Cons - Scattered permission control, too much power for the object's owner group.
3. MAC (Mandatory Access Control)
   Both resource owners and resources have classification labels. Different labels are granted with different permissions.
   Pros - strict and straightforward.
   Cons - not flexible.
4. ABAC (Attribute-based access control)
   Evaluate permissions based on attributes of the Resource owner, Action, Resource, and Environment.
   Pros - flexible
   Cons - the rules can be complicated, and the implementation is hard. It is not commonly used.
5. RBAC (Role-based Access Control)
   Evaluate permissions based on roles
   Pros - flexible in assigning roles.

Back-Of-The-Envelope Estimation / Capacity Planning



Org charts comic

by Manu Cornet



According to Wikipedia, the drawing appeared in The New York Times, and Microsoft CEO Satya Nadella cited that it was what persuaded him to change Microsoft's culture.

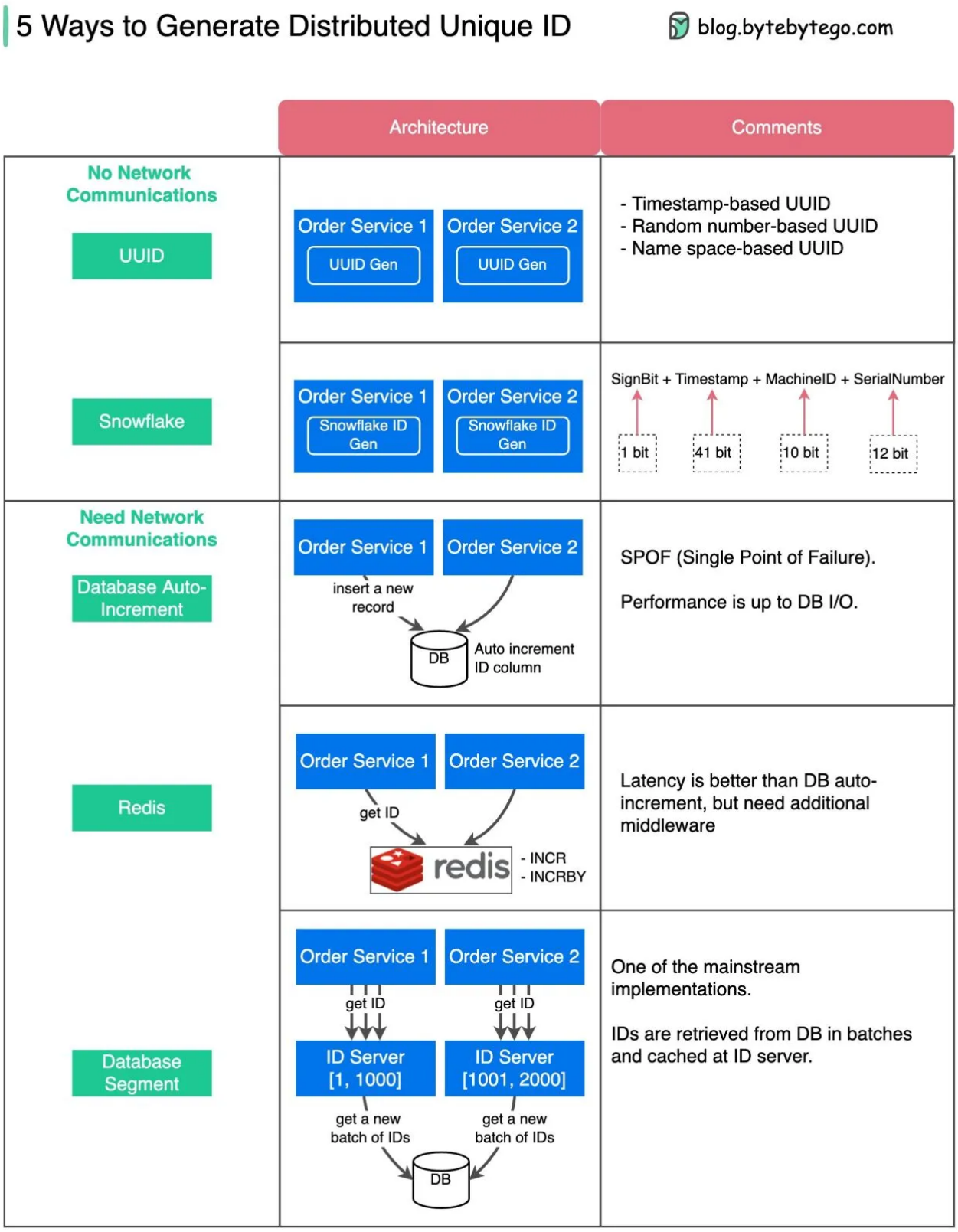
Link to the drawing: https://lnkd.in/eh7RiMKa

The drawing was published in 2011. More than 10 years have passed. How relevant is this now?

Five ways to generate distributed unique ID

How do we generate unique IDs in distributed systems? How do we avoid ID conflicts?

The diagram below shows 5 ways. 👇



Assume the design requirements of distributed unique ID are:

1. Globally unique.
2. Availability. The ID generator must be available under high concurrency.
3. Ordered. The IDs are sorted by certain rules. For example, sorted by time.
4. Distributed. The ID generator doesn't rely on a centralized service.
5. Security. Depending on the use case, some IDs cannot be just incremental integers, which might expose sensitive information. For example, people might guess the total user number correctly by looking at the sequence IDs.

Thanks for making it this far!

If you want to learn more about System Design, check out our books:

Paperback edition

Digital edition

Thanks for reading ByteByteGo Newsletter!
Subscribe for free to receive new posts and

support my work.

85 Likes

1 Comment

Write a comment...

Sanjeev MX Sep 19, 2022

Great newsletter!!

"Resiliency Patterns" section (https://blog.bytebytego.com/i/73735063/resiliency-patterns) - For like 10 seconds, I first thought left side was the problem and right side was the solution :) It confused me. Suggestion - mention 8 ways above the table and numbering each of them for readability.

LIKE (3) REPLY SHARE