Sikkerhet og tilgangskontroll i RDBMS-er

IN2090 – 14. nov 2018

Mathias Stang

Agenda

- Modeller for tilgangskontroll
- Brukere og roller i RDBMS-er
- GRANT og REVOKE
- SQL Injections

Hovedmål med databasesikkerhet

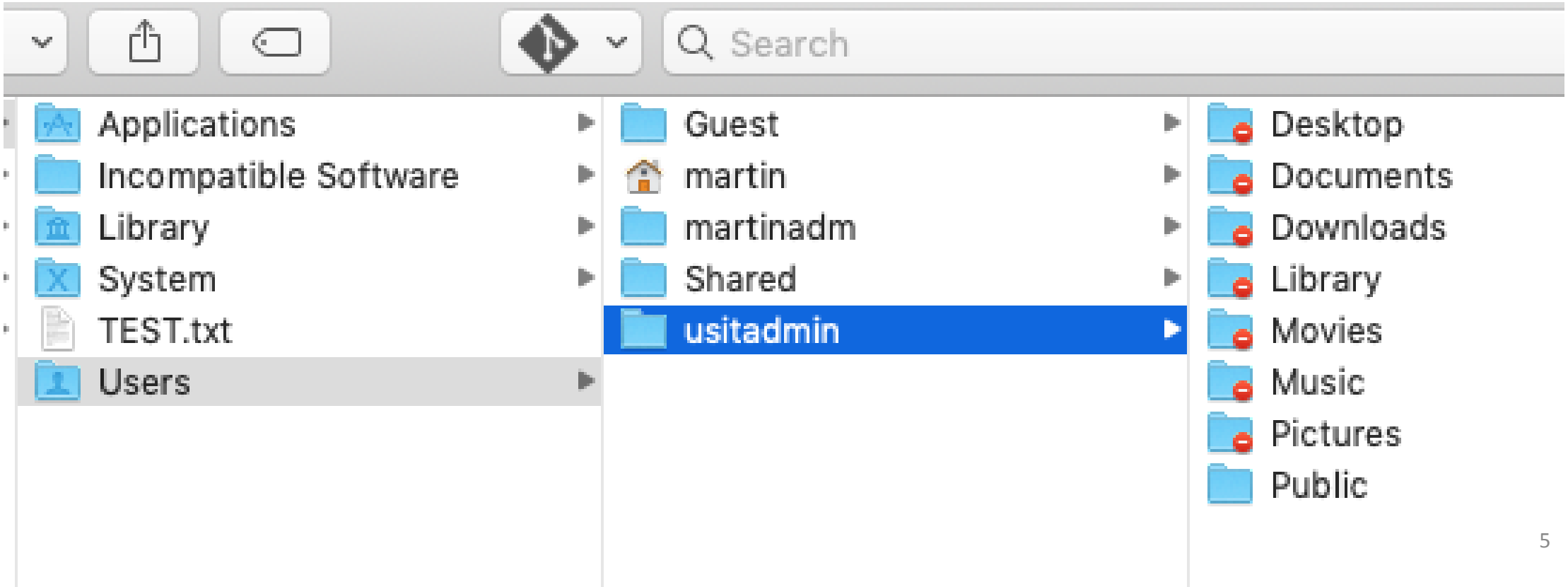
- **Konfidensialitet**
  - Uvedkommende må ikke kunne se data de ikke skal ha tilgang til.
- **Integritet**
  - Data må være korrekte og pålitelige. Derfor må data beskyttes mot endringer fra uautoriserte brukere.
- **Tilgjengelighet**
  - Brukere må kunne se eller modifisere data de har fått tilgang til

Sikkerhet: ikke bare i databasesystemet

- Databasesystemer eksisterer ikke isolert
- Sikkerhetshull kan finnes i alle ledd, f.eks. OS, nettverket, applikasjoner osv.
- Vi skal fokusere på tilgangskontroll i vanlige DBMS

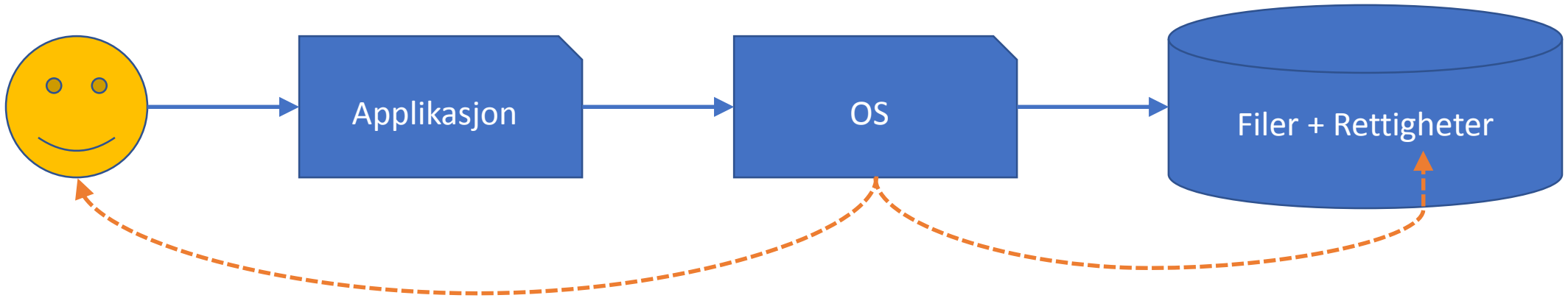
Tilgangskontroll for desktop-systemer

- Hvordan sikrer min Windows/Linux/macOS-maskin at jeg ikke leser andre brukeres filer på samme maskin?

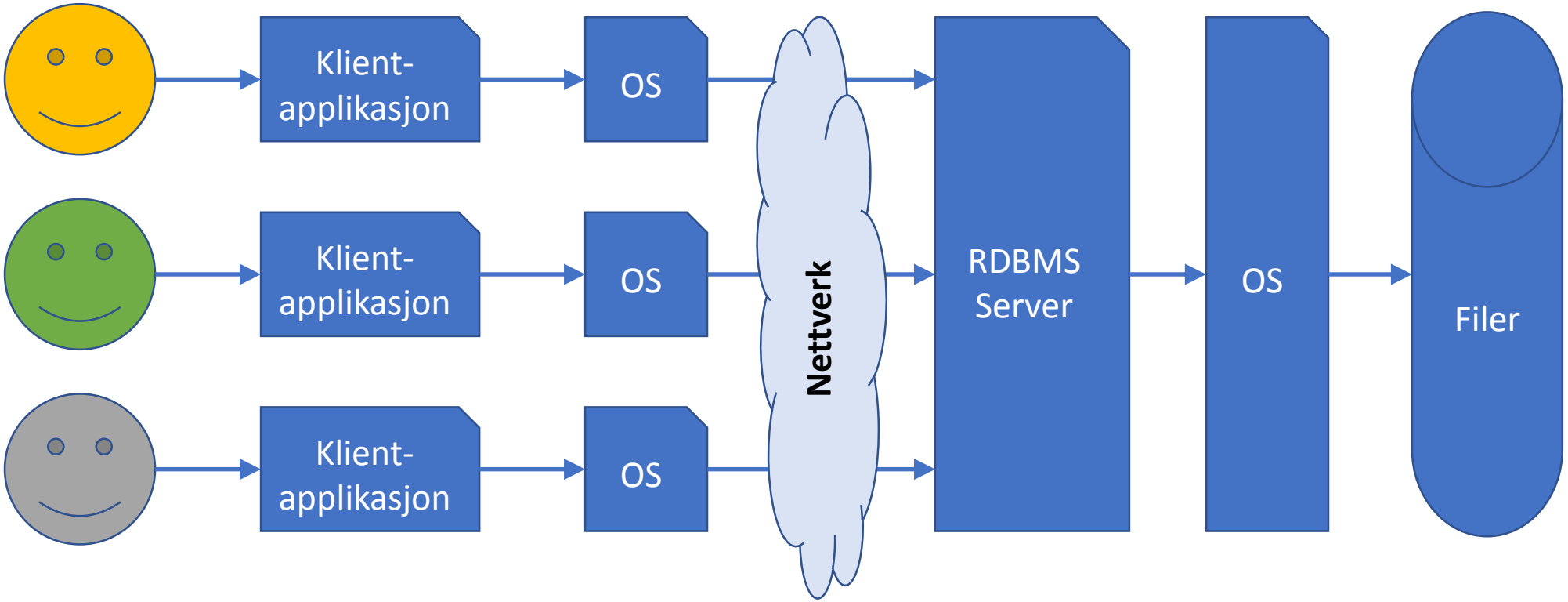


Autentisering for desktop-systemer

- Må logge seg inn først, med brukernavn og passord
- Tilgang til innhold avhengig av brukeren som er logget inn
- Filer osv. har ulike tilgangsrettigheter for ulike brukere
- OS kontrollerer at applikasjoner bare får tilgang til de riktige filene



Informasjonssystem med sentral RDBMS



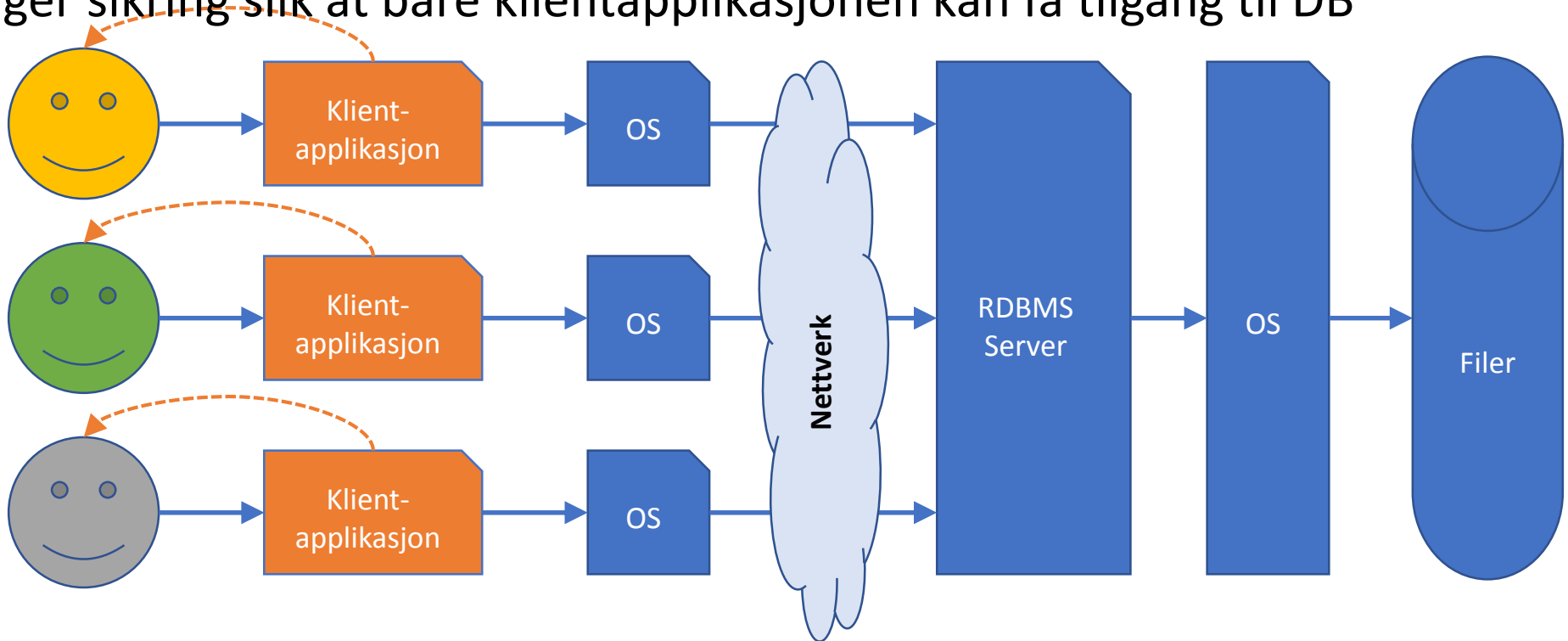
Tilgangskontroll med RDBMS

- OS-brukere på klienten ikke det samme som brukere for DB
  - Min laptop ser at jeg er logget inn som bruker «mathias»
  - Kan hvilken som helst database nå vite at jeg er Mathias J. Stang, som bor i Oslo, Norge…?

- To måter å styre brukertilgang på:
  - I klient- (eller backend-)applikasjonen
  - I databasesystemet

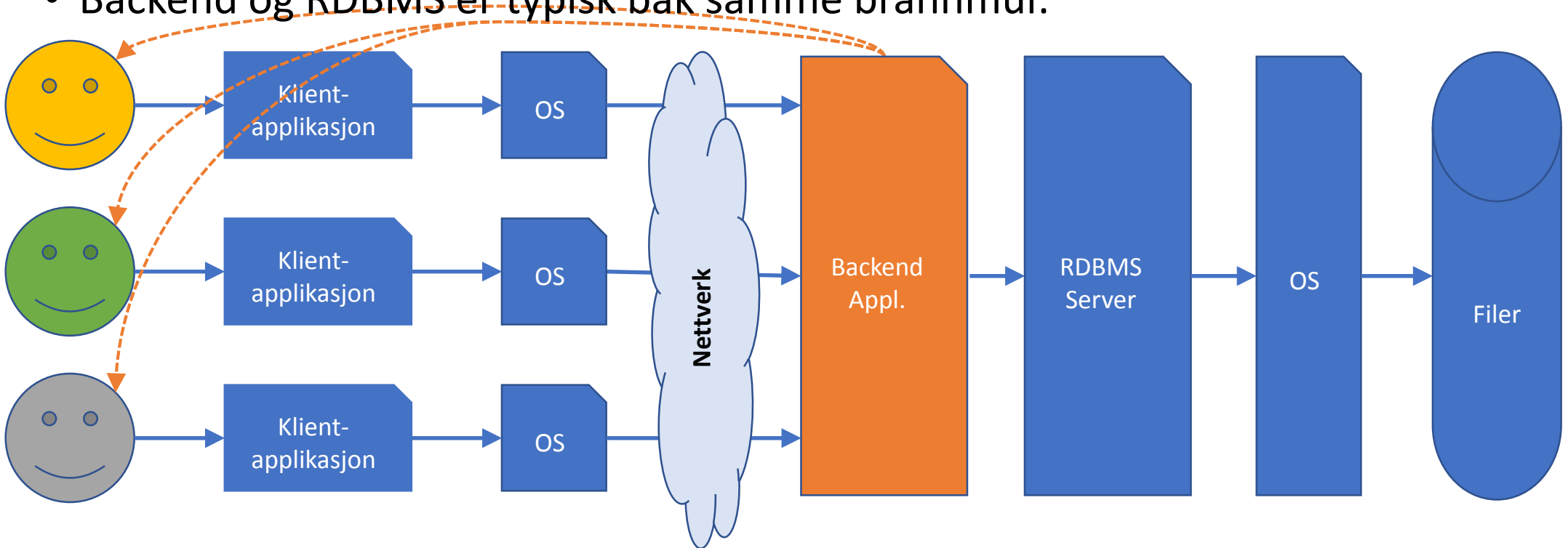
Tilgangskontroll i klientapplikasjonen

- Klienten autentiserer brukere
- Klienten sjekker hva brukeren får lov til å hente ut, legge inn, endre, slette…
- DB-systemet stoler på at klienten gjør dette riktig.
- Trenger sikring slik at bare klientapplikasjonen kan få tilgang til DB



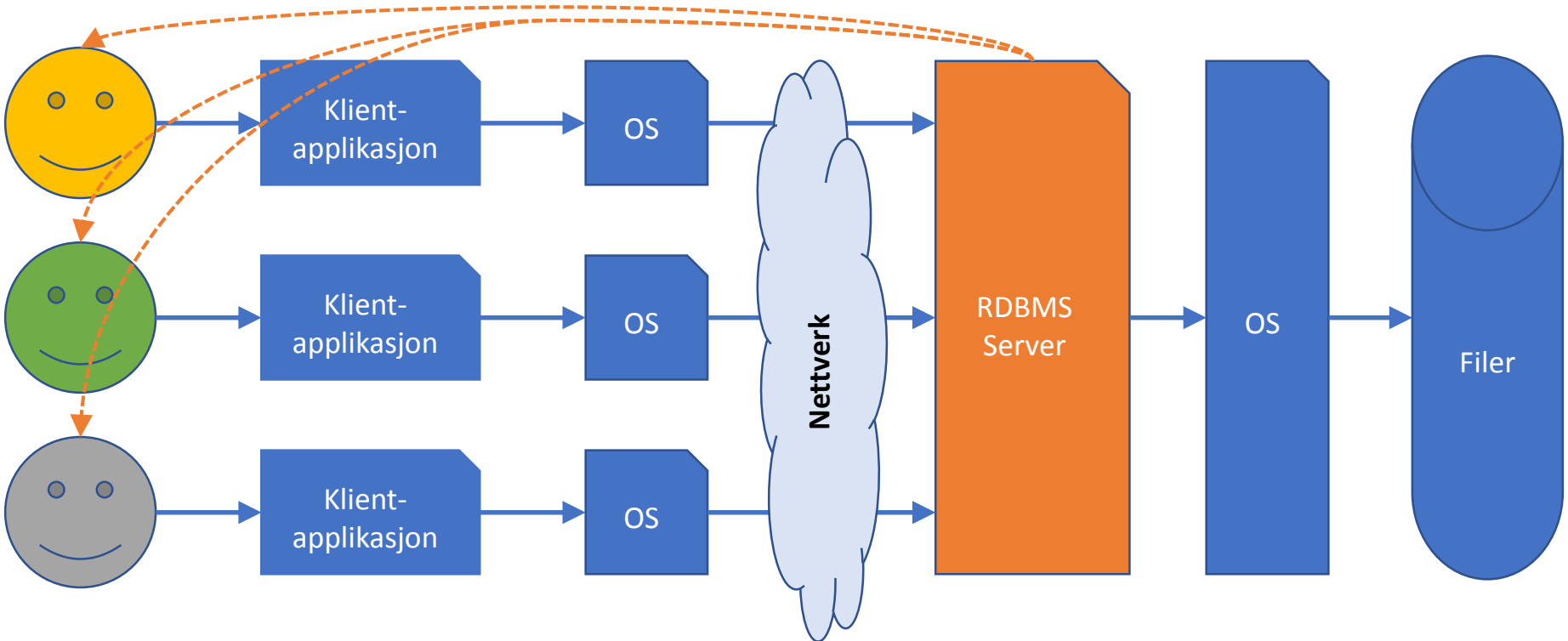
Tilgangskontroll i backend

- Klient-app håndterer autentisering i samspill med backend
- Backend sjekker hva brukere får lov til legge inn, endre, slette…
- DB systemet stoler på at backend gjør dette riktig.
- Backend og RDBMS er typisk bak samme brannmur.



Tilgangskontroll i RDBMS

- Klienten delegerer brukerautentisering til RDBMS
- RDBMS har endelig ansvar for hvilke brukere har hvilke rettigheter
- Klienten kan «forhåndssjekke»



Tilgangskontroll i SQL databaser

- Brukere (Users)
- Roller (Roles)
- Rettigheter (Permissions)
- For eksempel:
  - Bruker mathias har rollen kundeadmin
  - Bruker martin har rollen superbruker
  - Brukere med rollen brukeradmin kan legge til og slette kunder
  - Brukere med rollen brukeradmin kan *ikke* slette hele kundetabellen
  - Brukere med rollen superbruker kan slette hele kundetabellen

Brukere

- All interaksjon med databasesystemet er via en «bruker» (user)

```
psql -h dbpg-ifi-kurs -U mjstang -d fdb
```

- Autentisering: sjekke at brukeren er den den påstår å være.
  - Typisk med passord, SSH public keys, osv.
- Gyldige brukerkontoer og (krypterte) passord lagres av RDBMS.
- Databaser, skjemaer, tabeller, views, etc. «eies» av en bruker
  - Ingen umiddelbar kobling til rettigheter!
- Transaksjoner i databasen kan spores til brukeren som ba om dem
- SQL-kommando CREATE USER… bare for «superbrukere» på tjeneren

Roller

- Mulig å si at «bruker mjstang kan legge til og slette kunder»
- Men…
  - Vanskelig å holde oversikt over hvem som har hvilke rettigheter
  - Vanskelig å legge til en ny bruker? Har vi glemt noen rettigheter?
  - Vanskelig å endre «policy»
    - kundeadmin skal ikke kunne endre rabatt til en kunde, bare selgere kan det
- Role-based Access Control:
  - Gi rettigheter basert på roller brukere har
  - Lett å gi eller fjerne roller for en bruker
  - Lett å gi eller fjerne rettigheter for alle som innehar en rolle

Roller i SQL

- Lag en ny rolle «brukeradmin»:
  - `CREATE ROLE brukeradmin;`
- Gi rollen «brukeradmin» til brukeren «mjstang»
  - `GRANT brukeradmin TO mjstang;`
- Fjern rollen «brukeradmin» fra brukeren «mjstang»
  - `REVOKE brukeradmin FROM mjstang;`

Grunnleggende om RDBMS-tilgangskontroll

Generell (forenklet) syntaks:

```
GRANT {tillatelse} ON {objekt} TO {bruker_eller_rolle}
REVOKE {tillatelse} ON {objekt} FROM {bruker_eller_rolle}
```

- *Tillatelser* kan være en (eller flere, kommaseparert) av f.eks.:
  - `SELECT, INSERT, UPDATE, DELETE, CREATE, CONNECT, ALL`
- *Objekt* kan f.eks. være tabellnavn, eller «`DATABASE`»
  (f.eks. for `CREATE` eller `CONNECT`)
- Tillatelser kan gis til *brukere* eller *roller* – eller til `PUBLIC` (alle)

Eksempler på GRANT

```
GRANT SELECT ON tliste TO kundeadmin;
GRANT INSERT, DELETE, UPDATE ON tliste TO mjstang;
```

Vi kan også sette tillatelser på enkeltattributter:

```
GRANT UPDATE (beskrivelse) ON tlistelinje TO mjstang;
```

GRANT OPTION

- Hvis vi vil gi brukere tillatelse til å gi andre brukere tillatelser, legger vi til `WITH GRANT OPTION` på slutten:

```
GRANT SELECT ON tliste TO mjstang WITH GRANT OPTION
```

- Linjen ovenfor gir brukeren *mjstang* SELECT-tillatelse på tabellen tliste, samt tillatelsen til å gi andre brukere eller roller SELECT-tillatelse

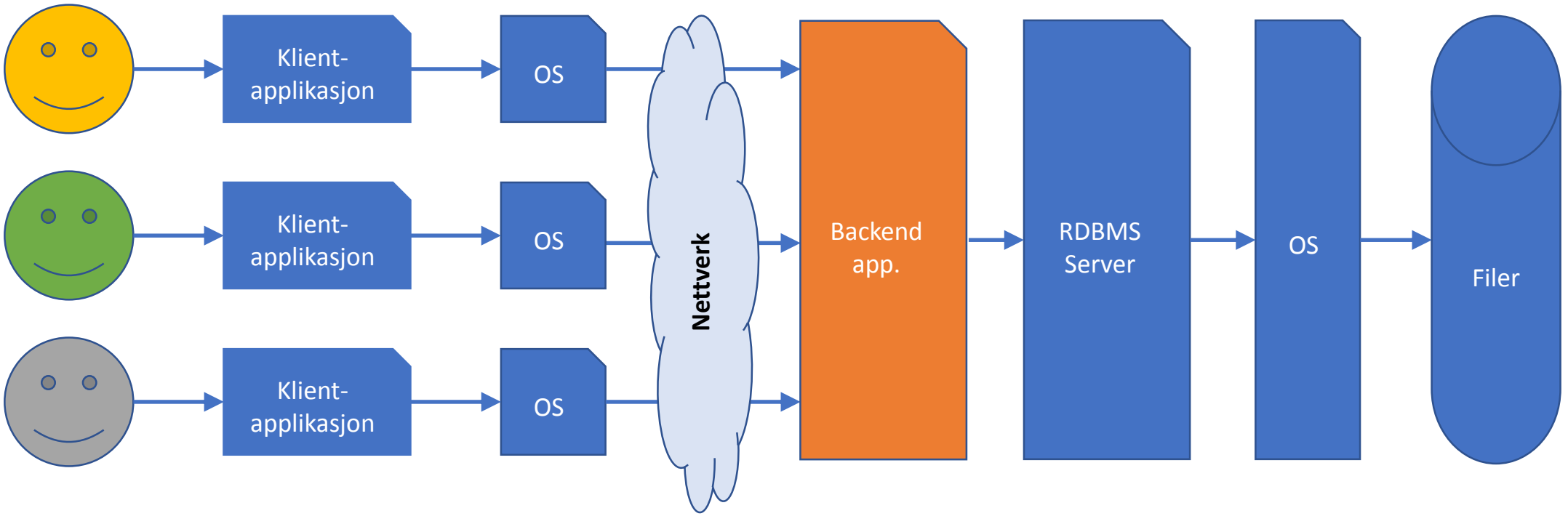
Views og tilgangskontroll

- I stedet for å gi tilganger til tabeller, kan man lage views som et «vindu» til tabellene, og gi tilganger til views i stedet

- Det er faktisk også mulig med «updatable views», men dette faller utenfor pensum

- Det finnes også flere mekanismer som lar oss styre hva brukere og roller har tillatelse til å gjøre – på radnivå – men det faller også utenfor pensum.

SQL Injection



SQL Injection

- Et type angrep mot en klientapplikasjon/backend der en (ondsinnet) bruker sender egen SQL-kode som så kjøres

- Da kan man f.eks. kjøre SELECT på andre ting enn utvikleren hadde tenkt, eller gjøre vilkårlige endringer i databasen

- Utnytter følgende faktorer:
  - At vi har et program som kjører SQL-spørringer (f.eks. Java eller Python)
  - Programmet tar input fra brukeren
  - SQL ikke har et skille mellom *data* og SQL-kommandoer

    → Der programmet skulle ha satt inn brukerdata i en spørring, blir det i stedet satt inn en SQL-kommando fra brukeren

SQL Injection: Grunnleggende prinsipp

En nettbutikk lar deg søke etter produkter slik:

```
String produktnavn = brukerinput();
String sql = "SELECT * FROM prod WHERE navn = '"+produktnavn+"'";
```

Med input «hei» får vi følgende spørring:

```
SELECT * FROM prod WHERE navn = 'hei';
```

Men med input «O'boy» får vi syntaksfeil:

```
SELECT * FROM prod WHERE navn = 'O'boy';
```

Hva med input «Hei'; DROP TABLE prod;--»?

```
SELECT * FROM prod WHERE navn = 'Hei'; DROP TABLE prod;--';
```

SQL injection

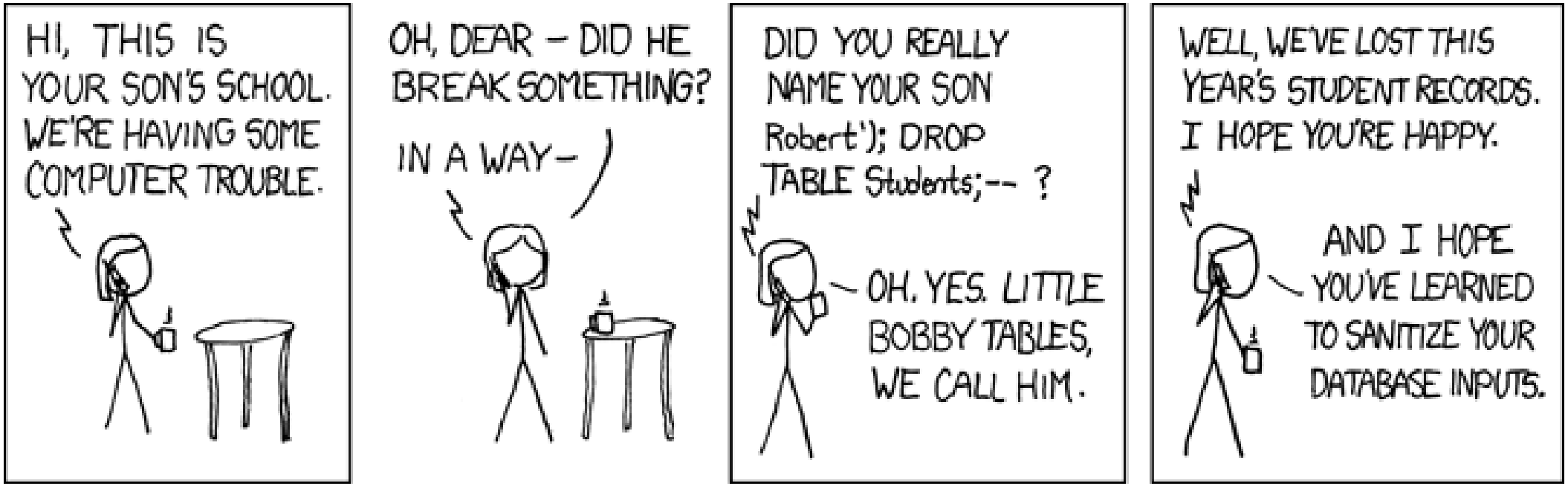
En liten demo – klarer vi å hacke denne tjenesten?

SQL Injection – hvorfor er dette viktig?

- Et av de vanligste formene for hackerangrep – lar ondsinnede brukere hente informasjon de ikke skulle hatt tilgang til, eller endre (eller ødelegge) data etter eget ønske

- Skyldes vanligvis at brukerdata ukritisk konkateneres inn i en spørring

- Løses veldig enkelt med å bruke parametriserte spørringer



https://xkcd.com/327/

Skjønte du den ikke?
https://stackoverflow.com/questions/332365/how-does-the-sql-injection-from-the-bobby-tables-xkcd-comic-work

PS: Husk repetisjonsforelesning på mandag!