Database Design: Boyce-Code & 3rd Normal Form

Abdu Alawini

University of Illinois at Urbana-Champaign

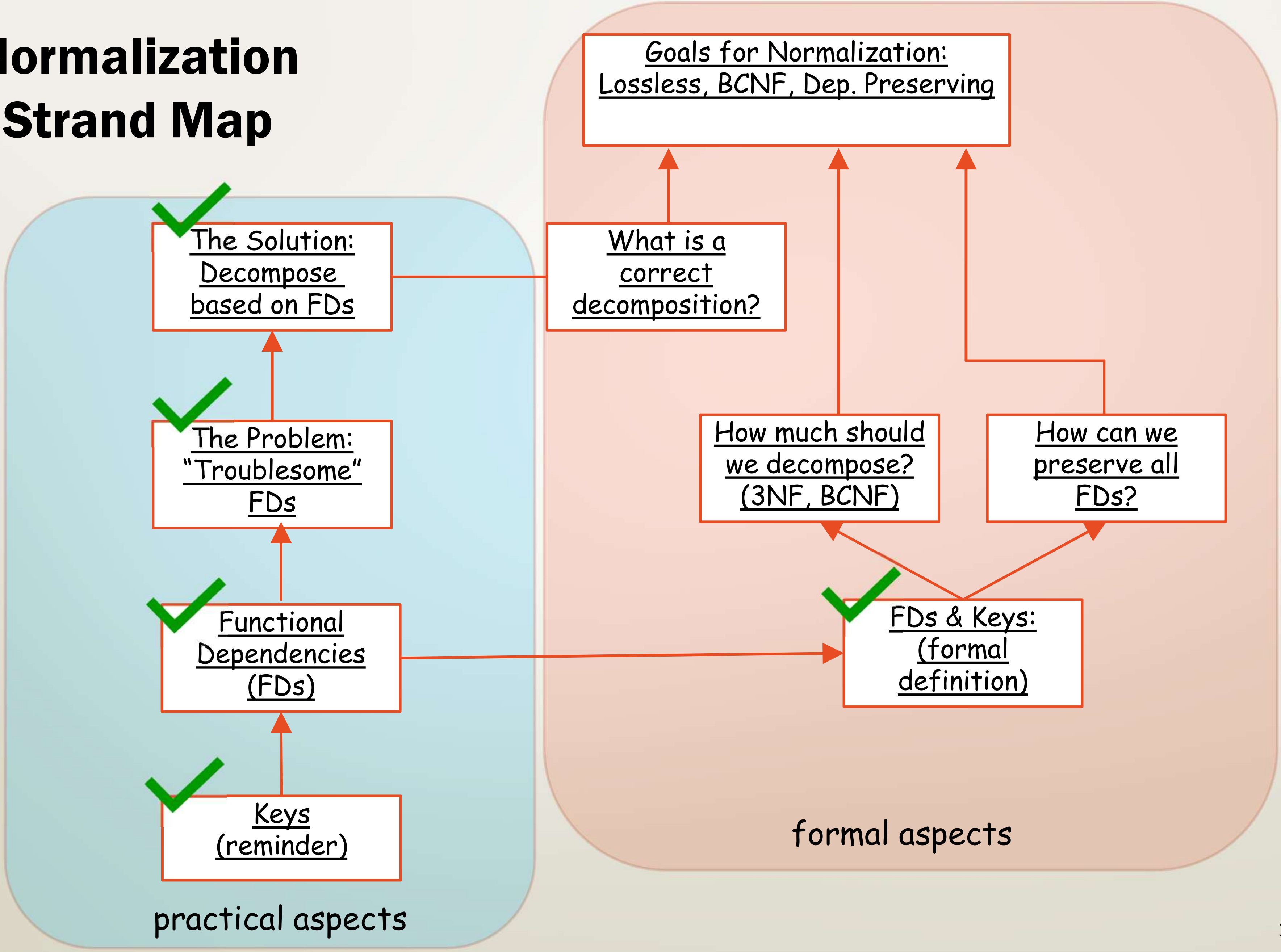
CS411: Database Systems

Learning Objectives

After this lecture, you should be able to:

- Decompose a database schema into a set of relations obeying BCNF
- Decompose a database schema into a set of relations obeying 3NF

Normalization Strand Map



Eliminating Anomalies

Main idea:

- $X \rightarrow A$ is OK, if X is a (super)key
- $X \rightarrow A$ is NOT OK, otherwise
  - Need to decompose the table, but how?

Boyce-Codd Normal Form (BCNF)

Normal Forms

First Normal Form = all attributes are atomic

Second Normal Form (2NF) = old and obsolete

Boyce Codd Normal Form (BCNF)

Third Normal Form (3NF)

Others...

Boyce-Codd Normal Form

Definition. A relation R is in BCNF if and only if:

Whenever there is a nontrivial FD: $A_1A_2\ldots A_n \rightarrow B$,

then $A_1A_2\ldots A_n$ is a superkey for R.

There are no "bad" FDs: whenever there is a nontrivial FD, its left side must be a superkey

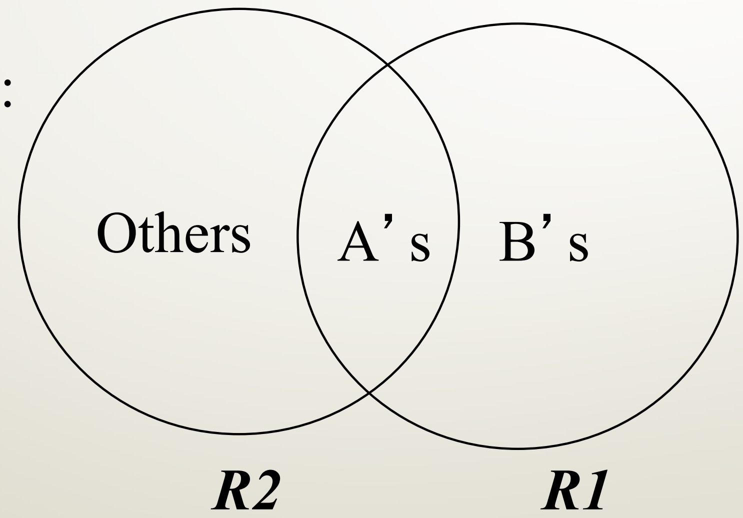
BCNF Decomposition

Find a dependency that violates the BCNF condition:

$$A_1, A_2, \ldots A_n \longrightarrow B_1, B_2, \ldots B_m$$

Heuristic : choose $B_1, B_2, \ldots B_m$ "as large as possible"

Decompose:



Continue until there are no BCNF violations left.

Example Decomposition

Person:

Name	SSN	Age	EyeColor	Phone

Functional dependencies:

SSN $\rightarrow$ Name, Age, Eye Color

BCNF: Person1(SSN, Name, Age, EyeColor),
Person2(SSN, Phone)

BCNF Decomposition: The Algorithm

Input: relation R, set S of FDs over R

1) Check if R is in BCNF, if not:

   a) pick a violation FD f: A → B

   b) compute A+

   c) create R1 = A+, R2 = A union (R – A+)

   d) compute all FDs over R1, using R and S. Repeat similarly for R2. (**See Algorithm 3.12**)

   e) Repeat Step 1 for R1 and R2

2) Stop when all relations are BCNF, or are two-attributes

(Two attribute relations are always in BCNF, see E.g. 3.17 (pg. 89) for proof and examples)

Another Example

- Person (Name, SSN, Age, EyeColor, Phone, HairColor)
- FD 1: SSN → Name, Age, EyeColor
- FD 2: Age → HairColor

FD 1 and 2 imply: SSN → Name, Age, EyeColor, HairColor

Iteration 1: Split based on SSN → Name, Age, EyeColor, HairColor

- Person(SSN, Name, Age, EyeColor, HairColor)
- Phone(SSN, Phone)

Iteration 2: Split based on Age → HairColor

- Person(SSN, Name, Age, EyeColor)
- Hair(Age, HairColor)
- Phone(SSN, Phone)

Q: Is BCNF Decomposition unique?

- R(SSN, netid, phone).
  - FD1: SSN -> netid
  - FD2: netid -> SSN
- Each of these two FDs violates BCNF.

Can you tell me two different BCNF decomp for R?

Pick FD1

R(SSN, netid, phone)

(SSN, netid) (SSN, phone)

Pick FD2

R(SSN, netid, phone)

(netid, SSN) (netid, phone)

Properties of BCNF

- BCNF removes certain types of redundancies
  - All redundancies based on FDs are removed.

- BCNF Decomposition avoids information loss
  - You can construct the original relation instance from the decomposed relations' instances.

How would get R(A, B, C) from R(A, B), R(B, C)?

A: Cross Product	B: Natural Join	C: Group BY

An easy decomposition?

Since two-attribute relations are always in BCNF.

Why don't we break any R(A,B,C,D,E)
into R1(A,B); R2(B,C); R3(C,D); R4(D,E)?

Why bother with finding BCNF violations etc.?

- *Turns out, this leads to information loss …*

Example of the "easy decomposition"

- R = (A,B,C); decomposed into R1(A,B); R2(B,C)

A	B	C
1	2	3
4	2	6

A	B
1	2
4	2

B	C
2	3
2	6

Example of the “easy decomposition”

- R = (A,B,C); decomposed into R1(A,B); R2(B,C)

A	B	C
1	2	3
4	2	6

A	B
1	2
4	2

B	C
2	3
2	6

Nat.Join

A	B	C
1	2	3
4	2	6
1	2	6
4	2	3

We get back some “bogus tuples” !

Lossless Decompositions

A decomposition is *lossless* if we can recover:

$$R(A,B,C) \xrightarrow{\text{Decompose}} \{\ R1(A,B)\ ,\ R2(A,C)\ \} \xrightarrow{\text{Recover}} R'(A,B,C) = R(A,B,C)$$

R' is in general larger than R. Why?

Must ensure R' = R

Desirable Properties of Schema Refinement

✓ 1) minimize redundancy

✓ 2) avoid info loss

3) preserve dependency

4) ensure good query performance

Normal Forms

First Normal Form = all attributes are atomic

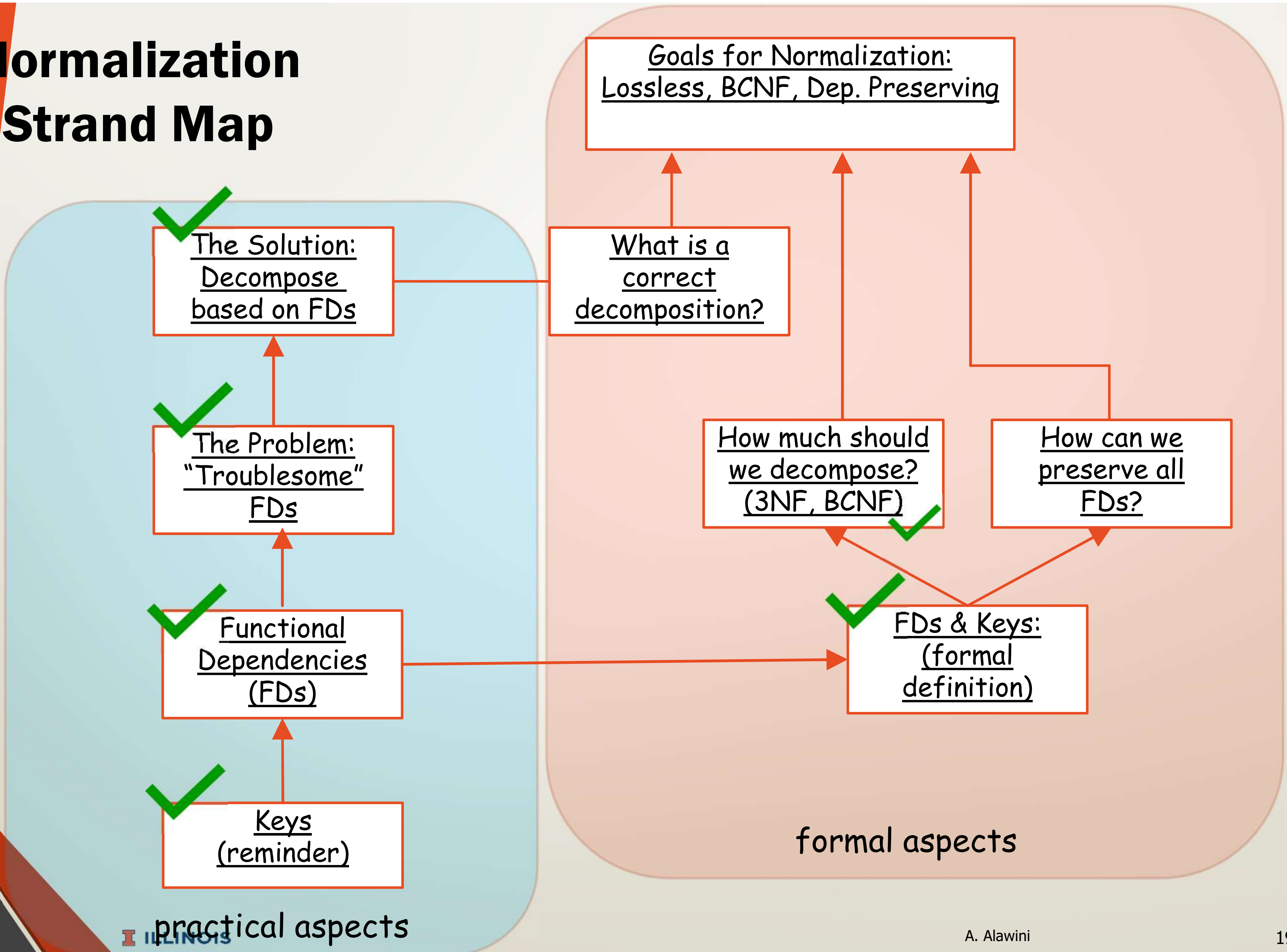
Second Normal Form (2NF) = old and obsolete

Boyce Codd Normal Form (BCNF)

Third Normal Form (3NF)

Others...

Normalization Strand Map



3NF: A Problem with BCNF

Phone	Address	Name

FD's: Phone → Address; Address, Name → Phone

So, there is a BCNF violation (Phone ➔ Address), and we decompose.

Phone	Address

Phone -> Address

Phone	Name

No FDs

So where's the problem?

Phone	Address
1234	10 Downing
5678	10 Downing

Phone	Name
1234	John
5678	John

FD's: Phone → Address; Address, Name → Phone

No problem so far. All *local* FD's are satisfied.

Let's put all the data into a single table:

Phone	Address	Name
1234	10 Downing	John
5678	10 Downing	John

Violates the dependency: Address, Name → Phone

Preserving FDs

- Thus, if the X and Y of a FD X->Y do not both end up in the same decomposed relation:
  - Such a decomposition is not "dependency-preserving."
  - No way to force BCNF to preserve dependencies
- Thus, while BCNF gives us **lossless join** and **less redundancy**, it doesn't give us **dependency preservation**

An alternative: 3rd Normal Form (3NF)

Definition. A relation R is in 3rd normal form if :

Whenever there is a nontrivial dependency $A_1, A_2, \ldots, A_n \rightarrow B$ for R, then $\{A_1, A_2, \ldots, A_n\}$ is a super-key for R, OR B is part of a key.

Prevents the "Phone ➔ Address" FD from causing a decomposition

Textbook uses rule with many B_i on the RHS, if so, then each one must be part of some key.

3NF vs. BCNF

- R is in BCNF if whenever X→ A holds, then X is a superkey.
  - Slightly stricter than 3NF.
    - Doesn't let R get away with it if A is part of some key
  - Thus, BCNF "more aggressive" in splitting
- Example: R(A,B,C) with AB → C; C → A
  - 3NF but not BCNF

Decomposing R into 3NF Preliminaries: Minimal basis

Given a set of FDs: F.

Say the set F' is *equivalent* to F, in the sense that F' can be inferred from F and v. versa.

- Any such F' is said to be a *basis* for F.

- "Minimal basis"
  - A basis with all RHS singletons, where any modifications lead to no longer a basis, including:
    - Dropping attribute from LHS of a rule: compact rules
    - Dropping a rule: small # of rules

Example of minimal basis

- R(A, B, C) with FDs:
  - A → BC; B → AC; C → AB
- A basis:
  - A→ B; A → C; B → A; B → C; C → A; C → B
- One minimal basis:
  - A → B
  - B → C
  - C → A

Conversion into minimal basis

- “Algorithm for converting F to a minimal basis
  - R = F with all RHS singletons:
  - Repeat until convergence:
    - If a rule minus an attribute from LHS is inferred from F, replace rule with rule minus attribute from LHS
    - If a rule is inferred from rest, drop it

Minimal basis example

Given R (A B C D E) and

F = { **A->D, BC->AD, C->B, E->A, E->D**}

Find F', the minimal basis for F.

Algorithm:

1- Only singleton in RHS

2- Remove unnecessary att. from LHS

3- Remove FDs that can be inferred from the rest

① $BC \rightarrow A$; $BC \rightarrow D$

② $BC \rightarrow A \Rightarrow C \rightarrow A$

$BC \rightarrow D \Rightarrow C \rightarrow D$

③ $F = \{A \rightarrow D, C \rightarrow A, \cancel{C \rightarrow D}, C \rightarrow B, E \rightarrow A, \cancel{E \rightarrow D}\}$

$E^+ = \{E, A, D\}$

$E \rightarrow D$ can be inferred

$F' = \{A \rightarrow D, C \rightarrow A, C \rightarrow B, E \rightarrow A\}$

Decomposing R into 3NF

1. Get a "minimal basis" *G* of given FDs
2. For each FD A ➔ B in the minimal basis *G*,
   use AB as the schema of a new relation.
3. If none of the schemas from Step 2 is a superkey,
   add another relation whose schema is a key for the original relation.

Result will be lossless, will be dependency-preserving, 3NF; might not be BCNF

Decomposing R into 3NF

1. Get a "minimal basis" G of given FDs
2. For each FD A → B in the minimal basis G,
   use AB as the schema of a new relation.
3. If none of the schemas from Step 2 is a superkey,
   add another relation whose schema is a key for the original relation.

Implicitly this is connecting all the LHSs with the remaining attributes

Result will be lossless, will be dependency-preserving,

Basically every minimal FD is preserved somewhere

Example

- R(A, B, C) with FDs:
  - A → BC; B → AC; C → AB

Minimal Basis: A → B; B → C; C → A

So, first cut:

R1(A, B), R2(B, C), R3(C, A)

Any attributes left? Nope → done

Example

- R(A, B, C, D, E) with FDs:
  - A → B; CD → B; DA → C

  BCNF Decomp:

  (AB), (ACD), (ADE) or:

  (BCD), (ACD), (ADE)

  Which FDs do each of these not preserve?

Minimal Basis:

A → B; CD → B; DA → C

3NF Decomp: (AB), (BCD), (ACD), (**ADE**)

Desirable Properties of Schema Refinement

1) minimize redundancy

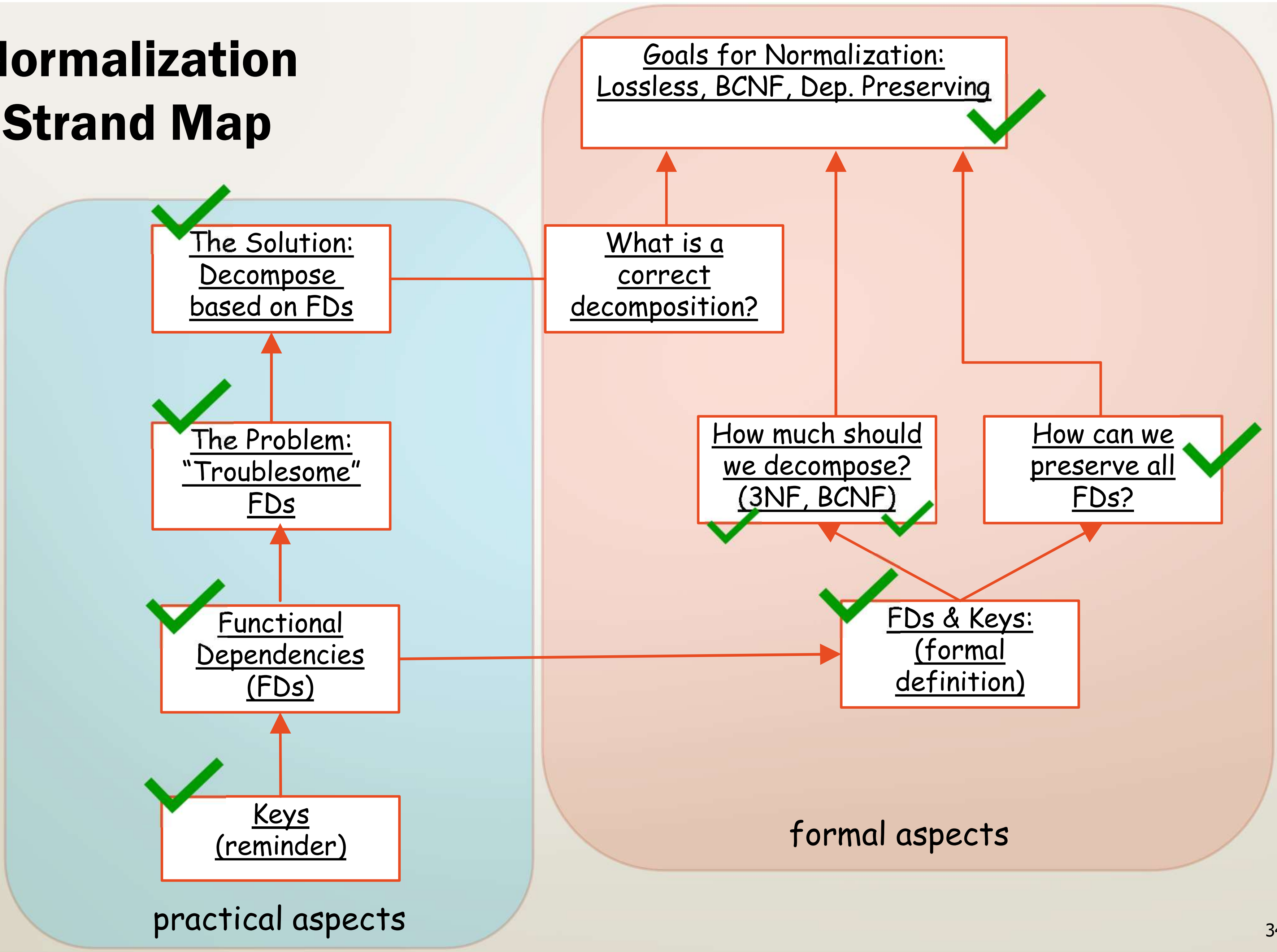
✓ **2) avoid info loss**

✓ **3) preserve dependency**

3NF

4) ensure good query performance

Normalization Strand Map



Caveat

- Normalization is not the be-all and end-all of DB design
- Example: suppose attributes A and B are always used together, but normalization theory says they should be in different tables.
  - decomposition might produce unacceptable performance loss (extra disk reads)

Overview of Database Design

- Conceptual design: (ER & UML Models are used for this.)
  - What are the **entities** and **relationships** we need?
- Logical design:
  - Transform ER design to Relational Schema
- Schema Refinement: (Normalization)
  - Check relational schema for redundancies and related anomalies.
- Physical Database Design and Tuning:
  - Consider typical workloads; (sometimes) modify the database design; select file types and indexes.

We'll discuss indexing next.