



Explaining Pair Programming Session Dynamics from Knowledge Gaps

Franz Zieris

zieris@inf.fu-berlin.de

Lutz Prechelt

prechelt@inf.fu-berlin.de

Motivation

- Expectations in industry:

Why pair-program?



Better design and
fewer defects

Learn from each other or together



Understand legacy parts of the
software system

- Our Overall Research Goal

- Understand how industrial pair programming actually works

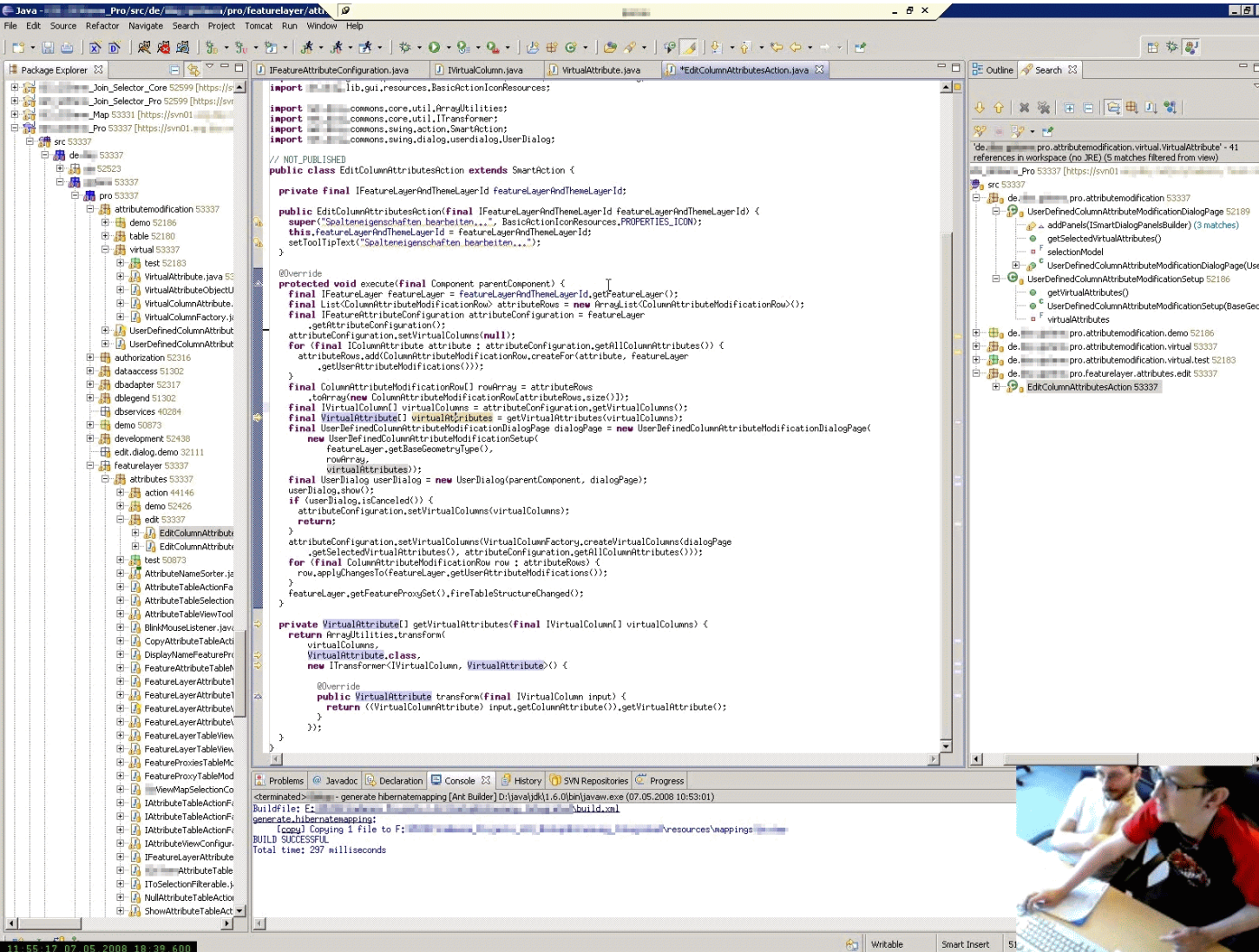
- Research Question

What are the underlying mechanisms of **knowledge transfer** in pair programming?

- Intended Outcome

- Advise practitioners
- behavioral (anti-)patterns

Qualitative Data Analysis



- Grounded Theory approach
- Recorded industrial PP sessions (audio, webcam, screen)

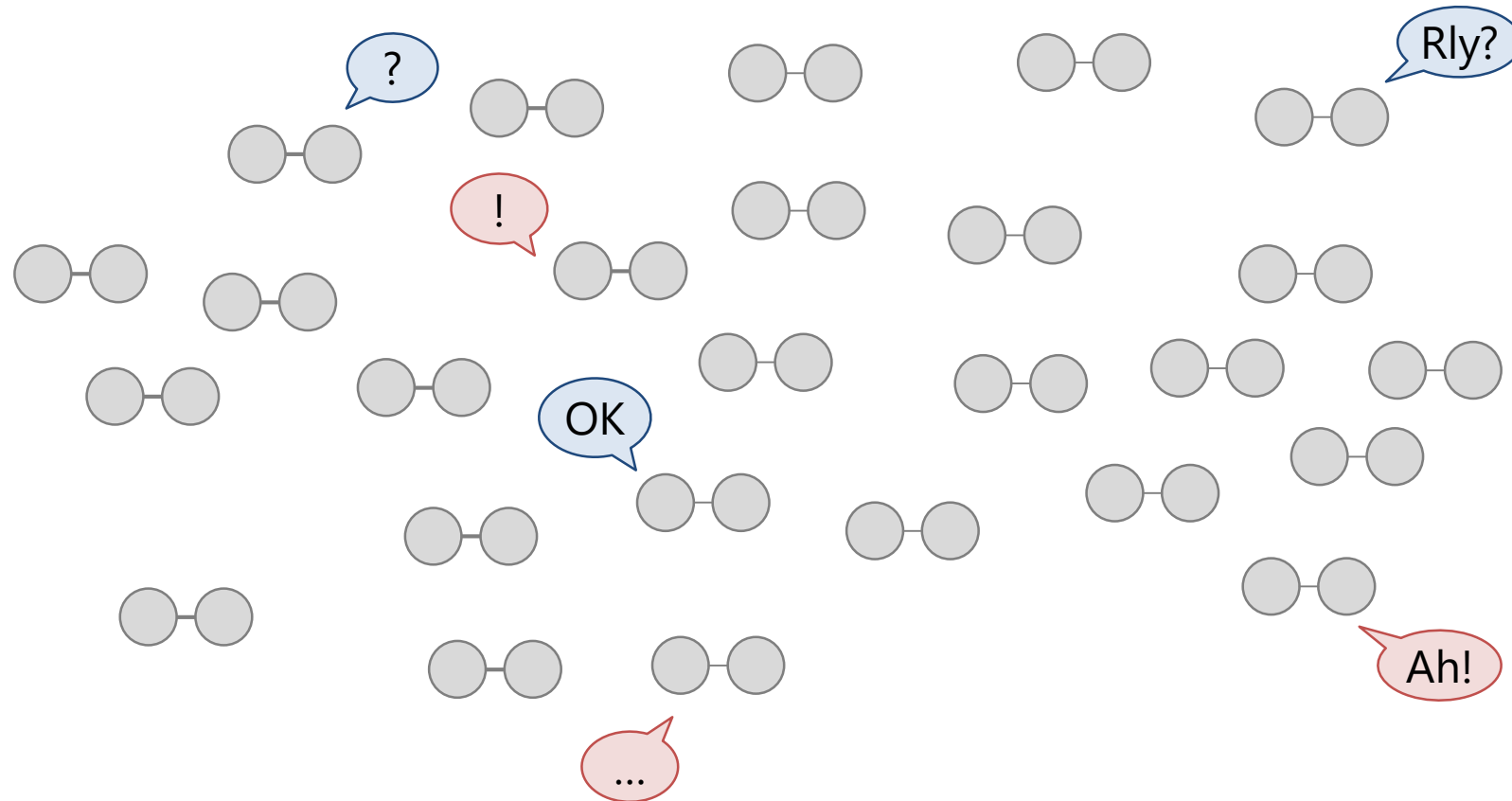


Theoretical sampling: 26 sessions (9 companies, 16 pairs)

- from a total of 67 sessions

Qualitative Data Analysis

26 pairings of
professional
developers

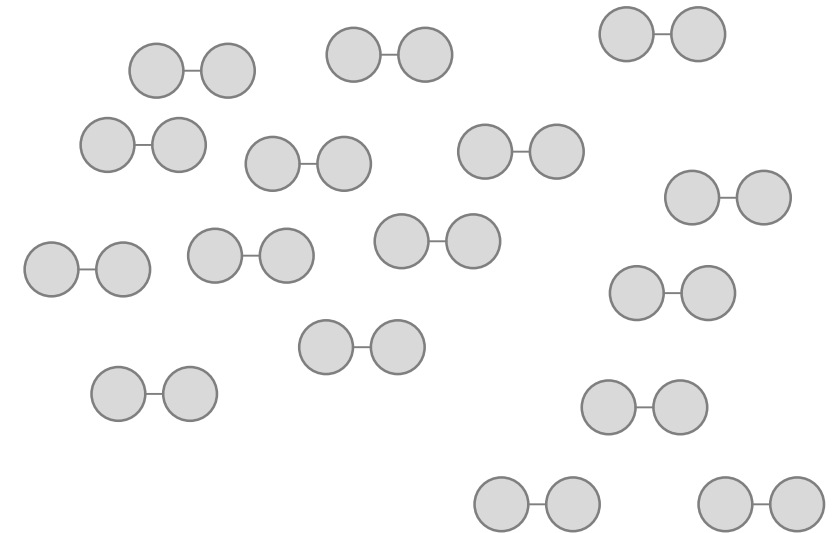
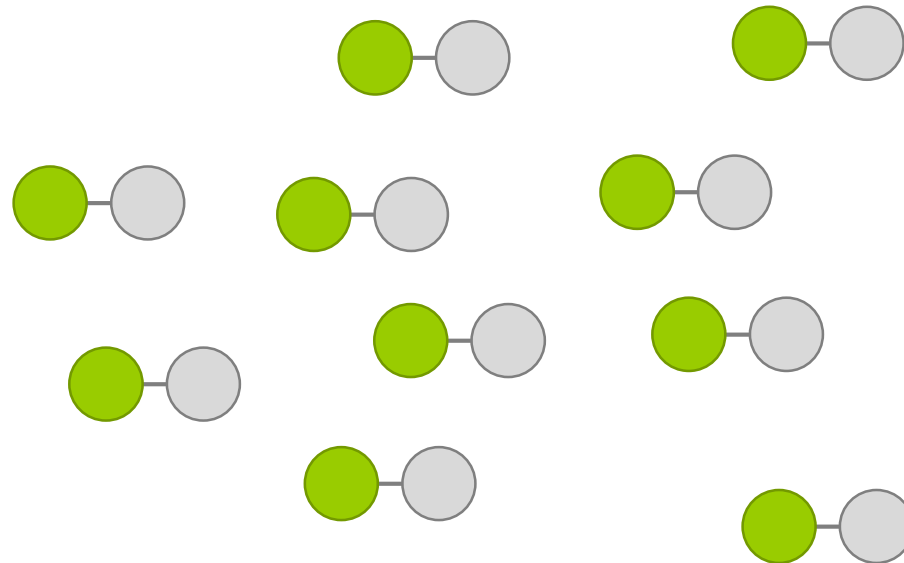


Analysis of pair programmers' dialog:

- What do they ask for? What do they explain?
- What do they know? What do they learn?

Observation 1: The Primary Gap

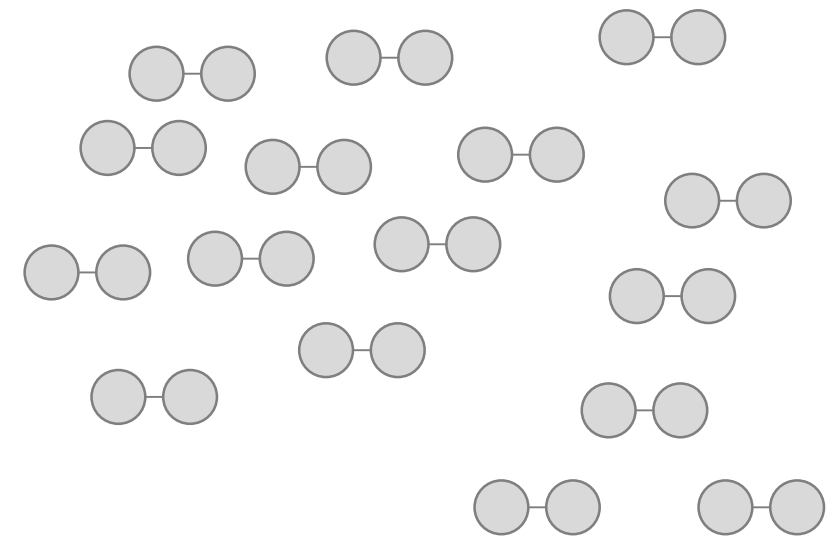
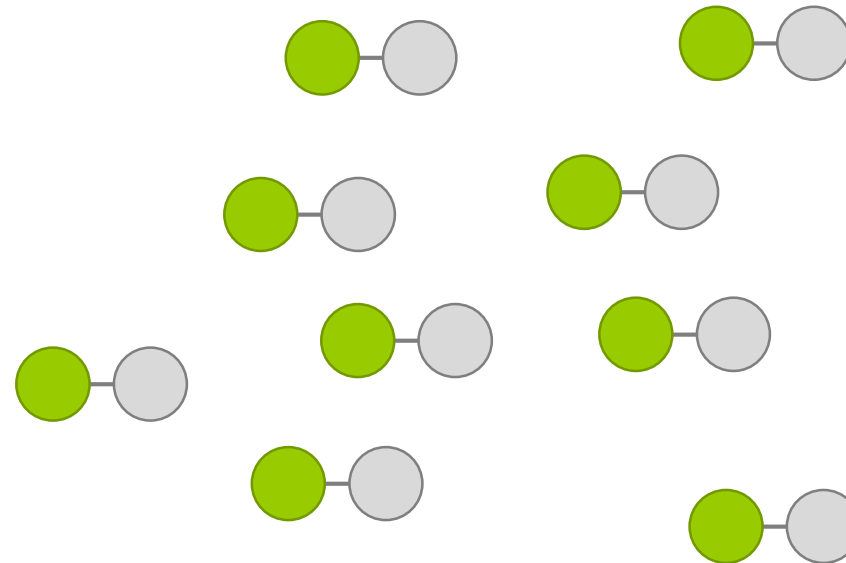
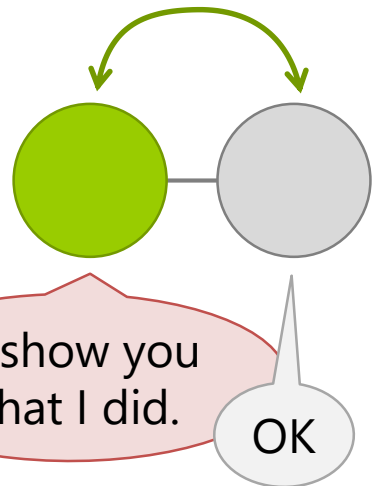
One partner already
worked on the task




Observation 1: The Primary Gap

One partner already worked on the task

Primary Gap

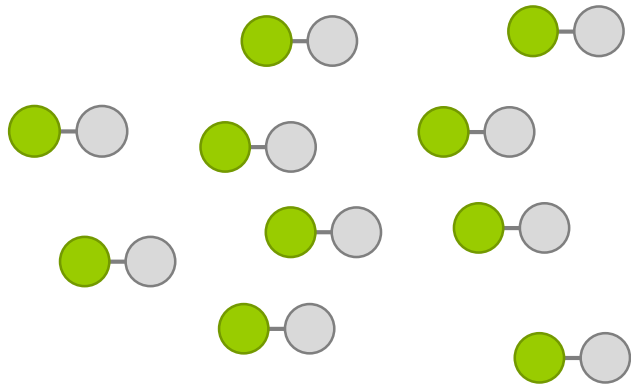


 = task-relevant system knowledge

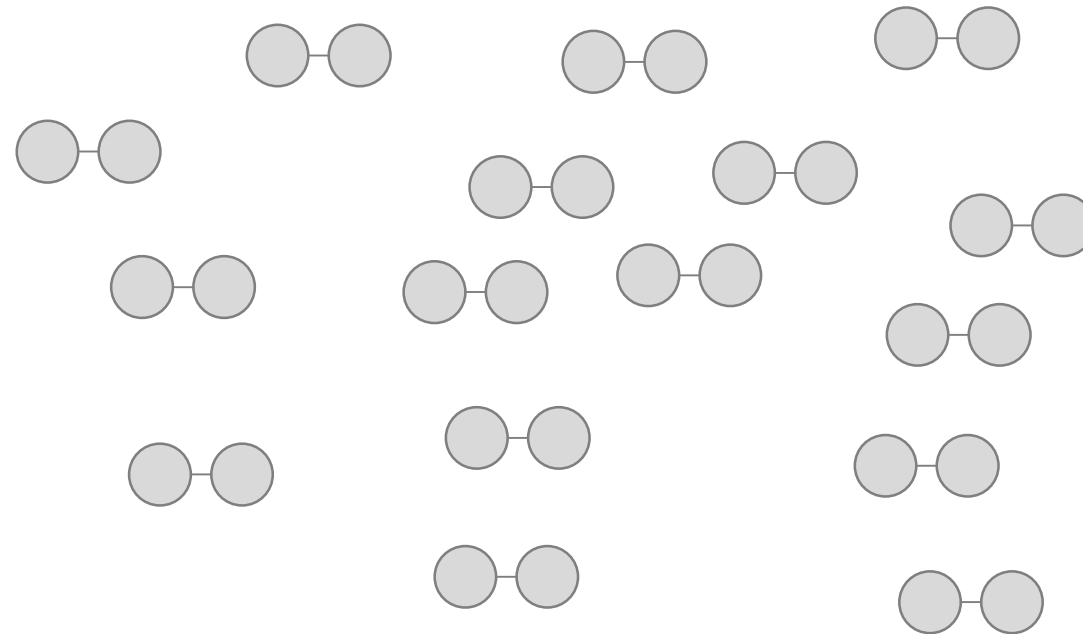
current state of implementation, classes, call hierarchies, defects, test/build setup, configuration state, ...

What about more homogenous pairs?

One partner already worked on the task



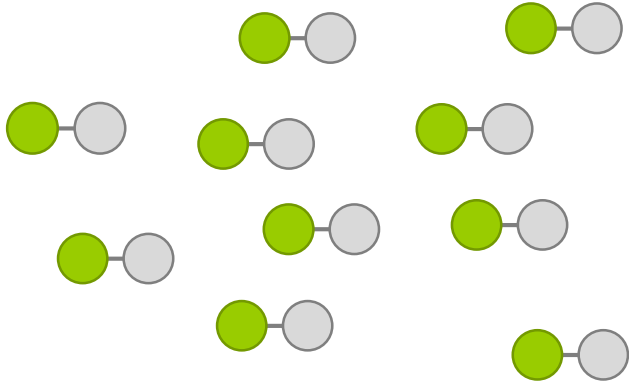
Both partners with similar prior involvement



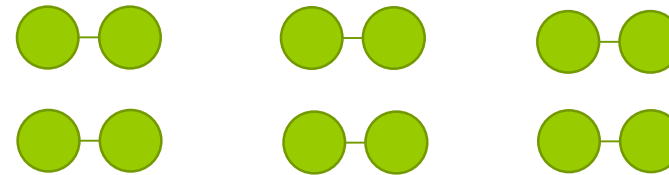
 = task-relevant system knowledge

What about more homogenous pairs?

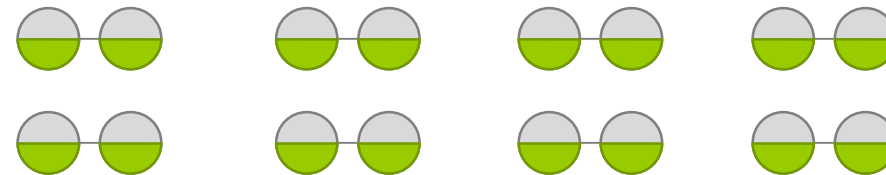
One partner already worked on the task



Both partners with similar prior involvement



recently worked in code area



basic knowledge of software

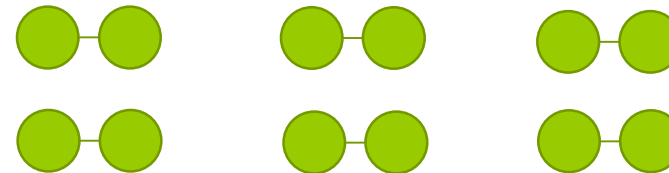


working in unknown terrain

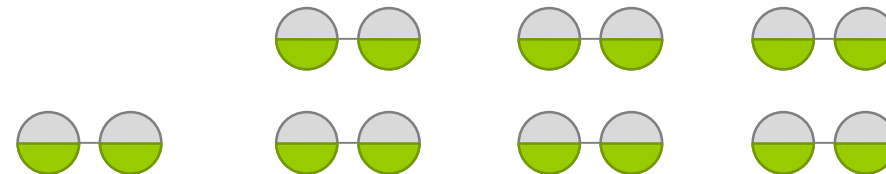
 = task-relevant system knowledge

Observation 2: The Secondary Gap

Both partners with similar prior involvement



recently worked
in code area



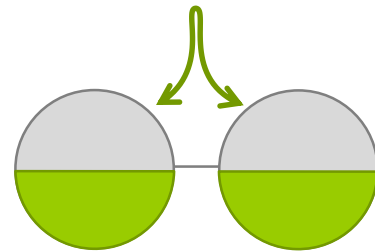
basic knowledge
of software



working in
unknown terrain

 = task-relevant system knowledge

Secondary Gap

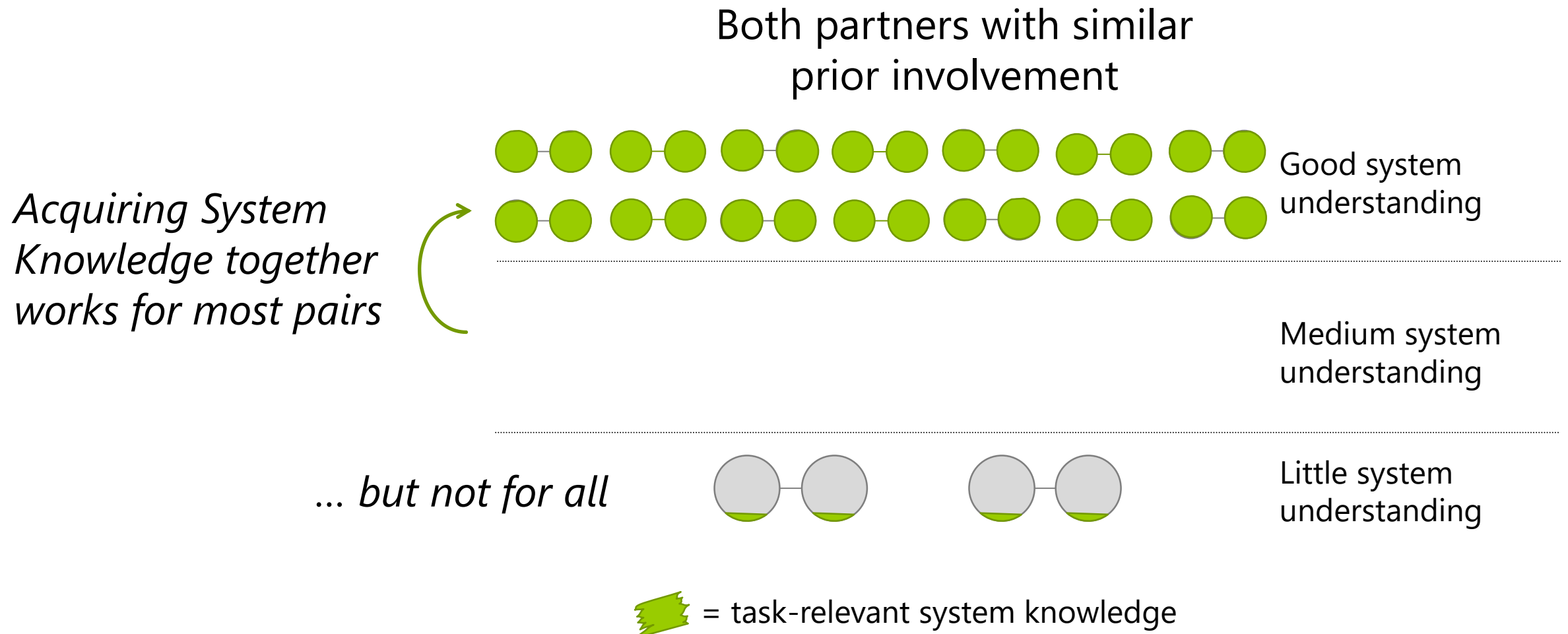


Let's look at
the superclass

`console.log(myObj);`



Observation 2: The Secondary Gap



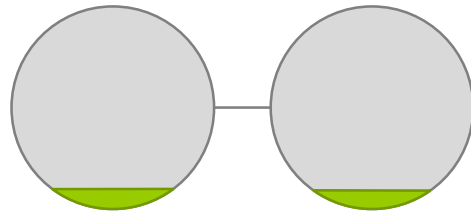
A Different Kind of Knowledge

Task: implement test case



Where is the initial value?


Type? Function?
I don't even know what this is.



 **S knowledge** (task-relevant system understanding)

Data structure holding the application state?
How to modify and read the state?

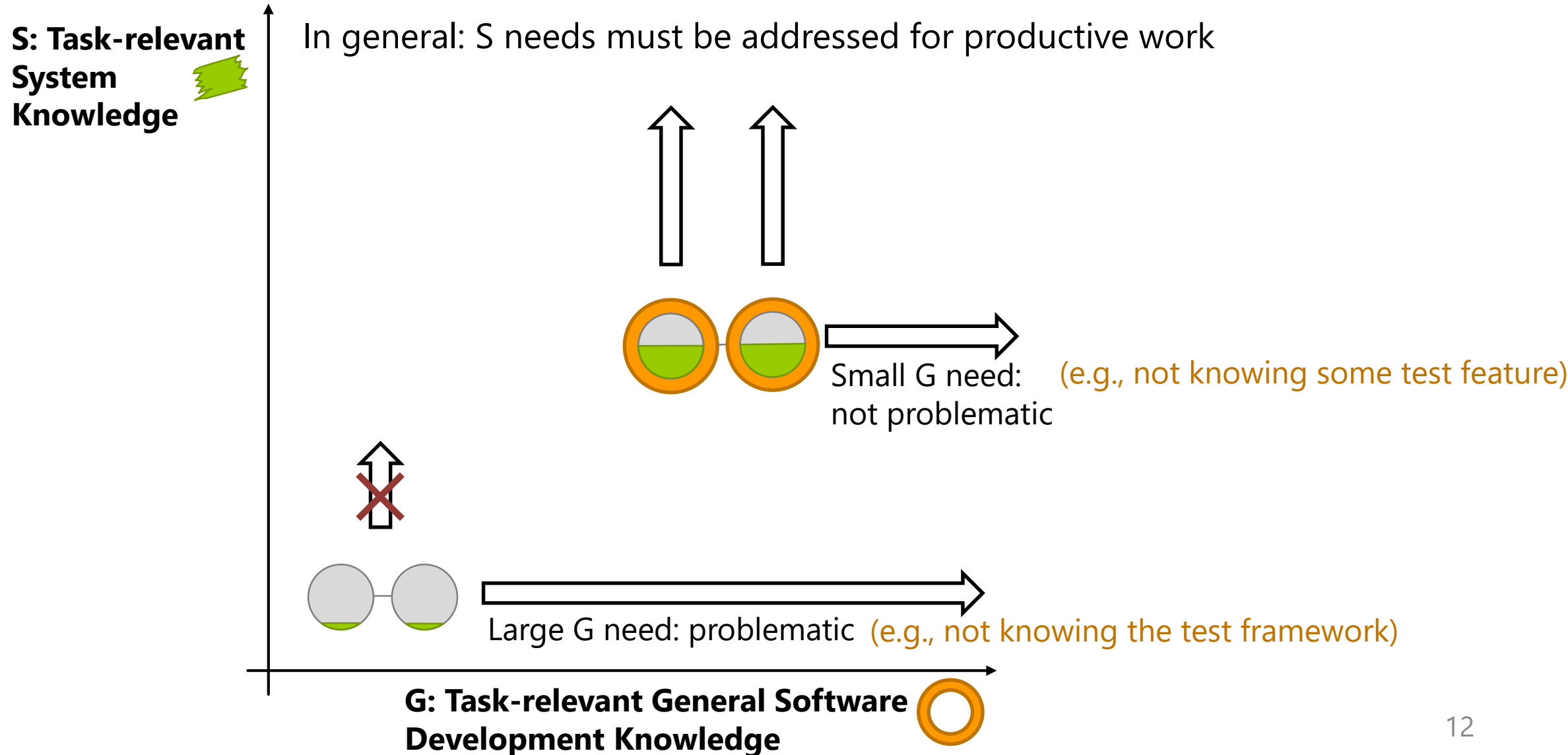
...

 **G knowledge** (task-relevant general software development knowledge)


Syntax of programming language?
Higher-order functions?
Application framework?
Test framework?

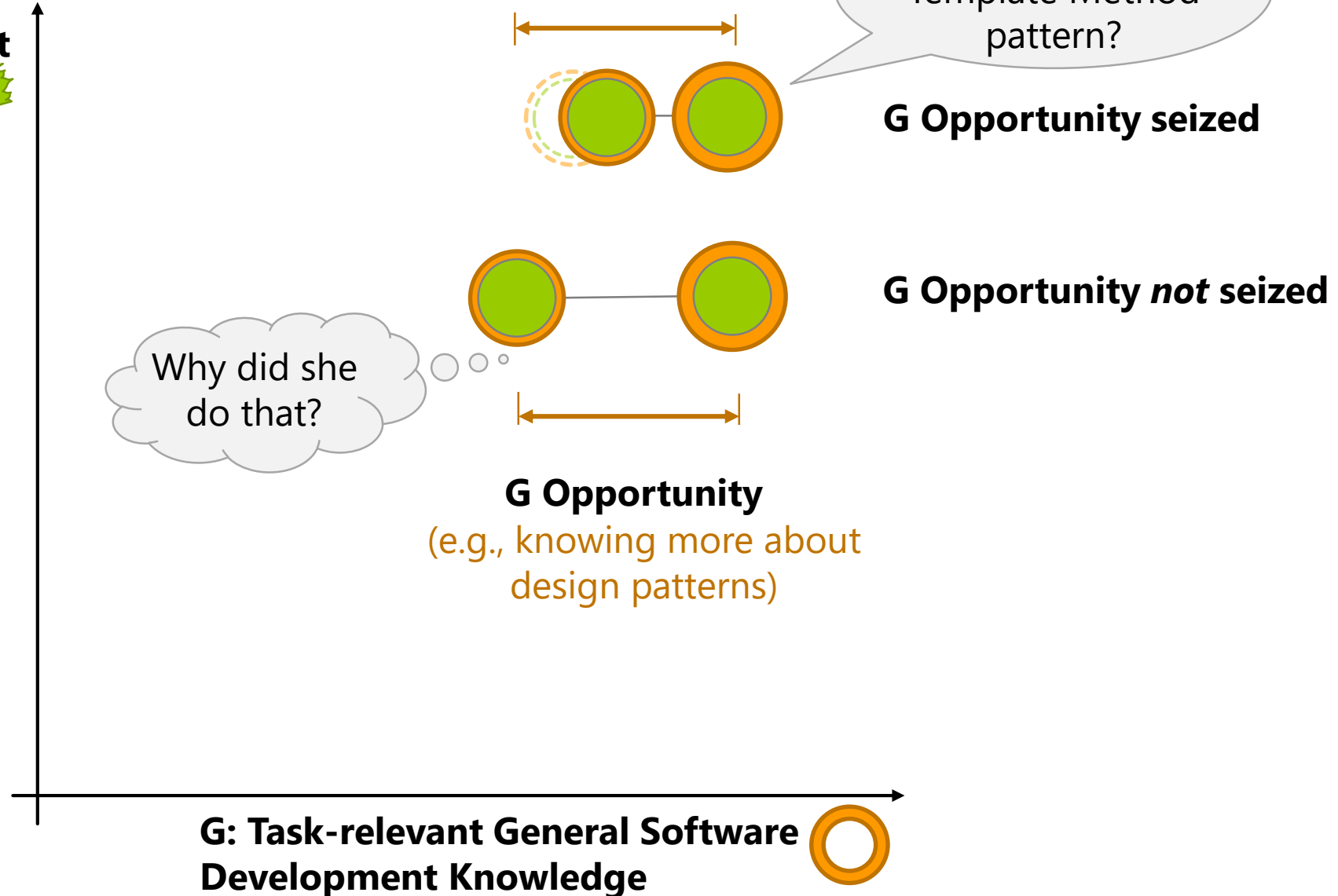
...

Roles of S and G knowledge

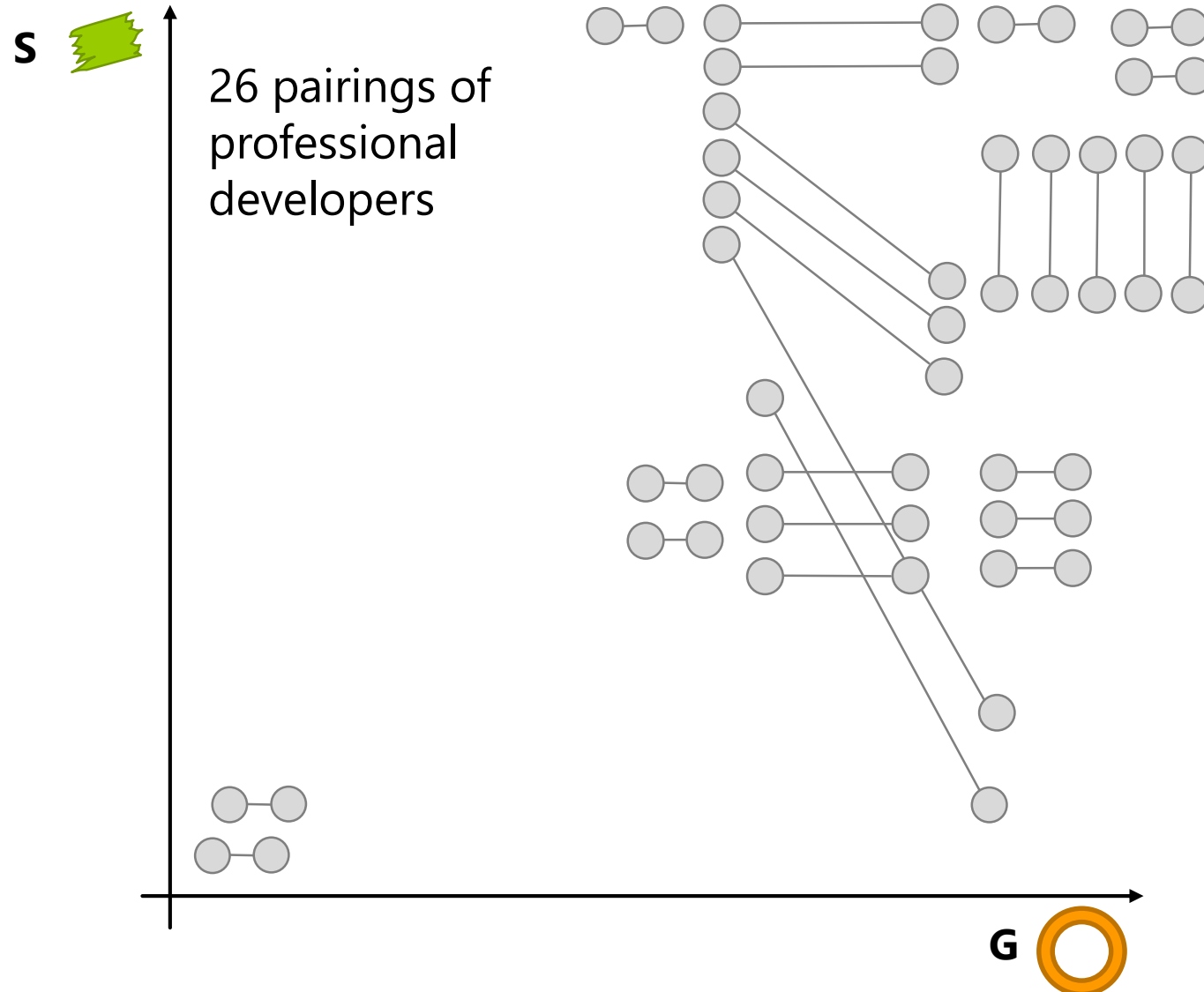


The G Opportunity

**S: Task-relevant
System
Knowledge** 



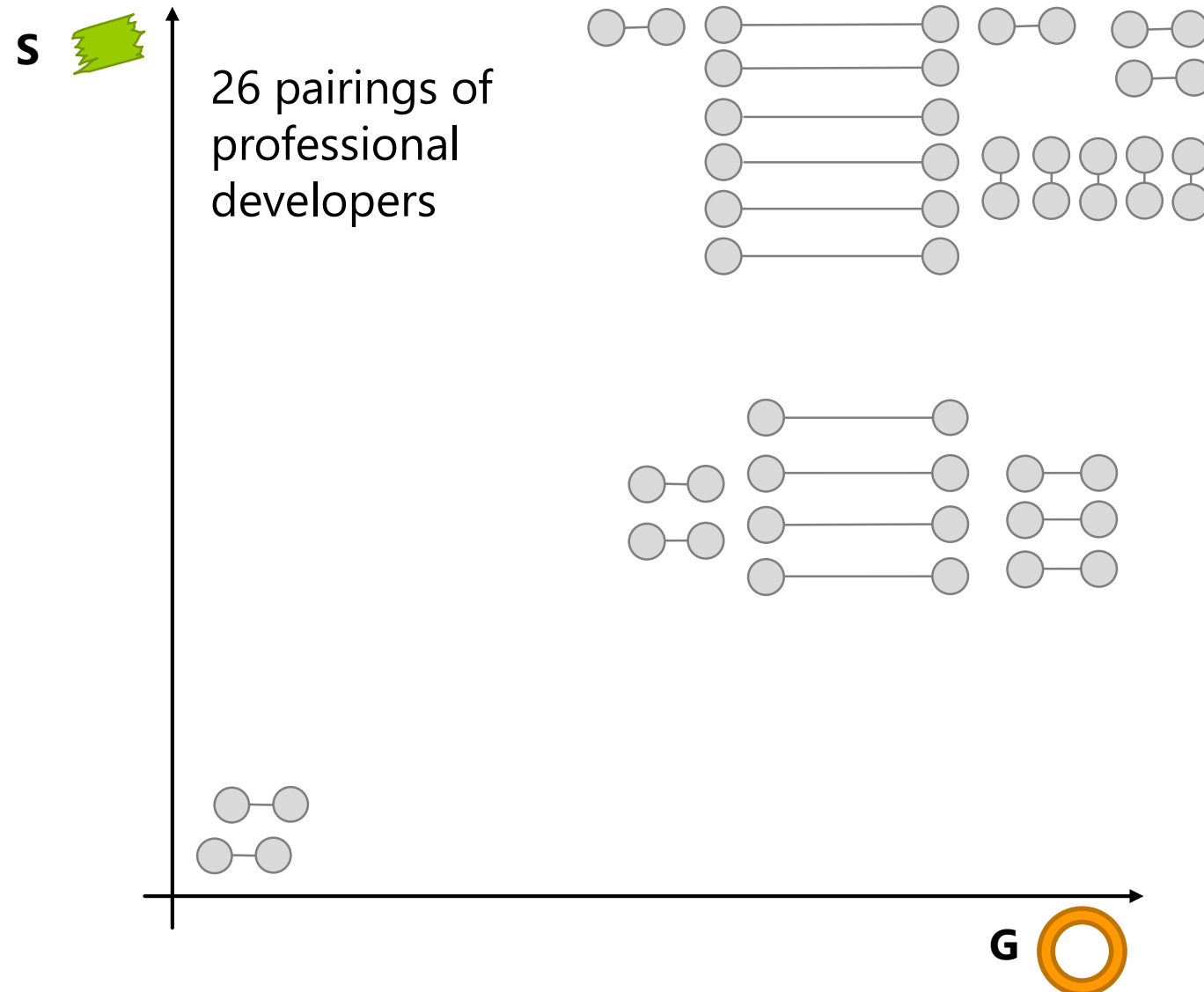
Overall Session Dynamic



Overall Dynamic

1. Close **Primary Gap**
2. Close **Secondary Gap**
3. Seize **G Opportunity**

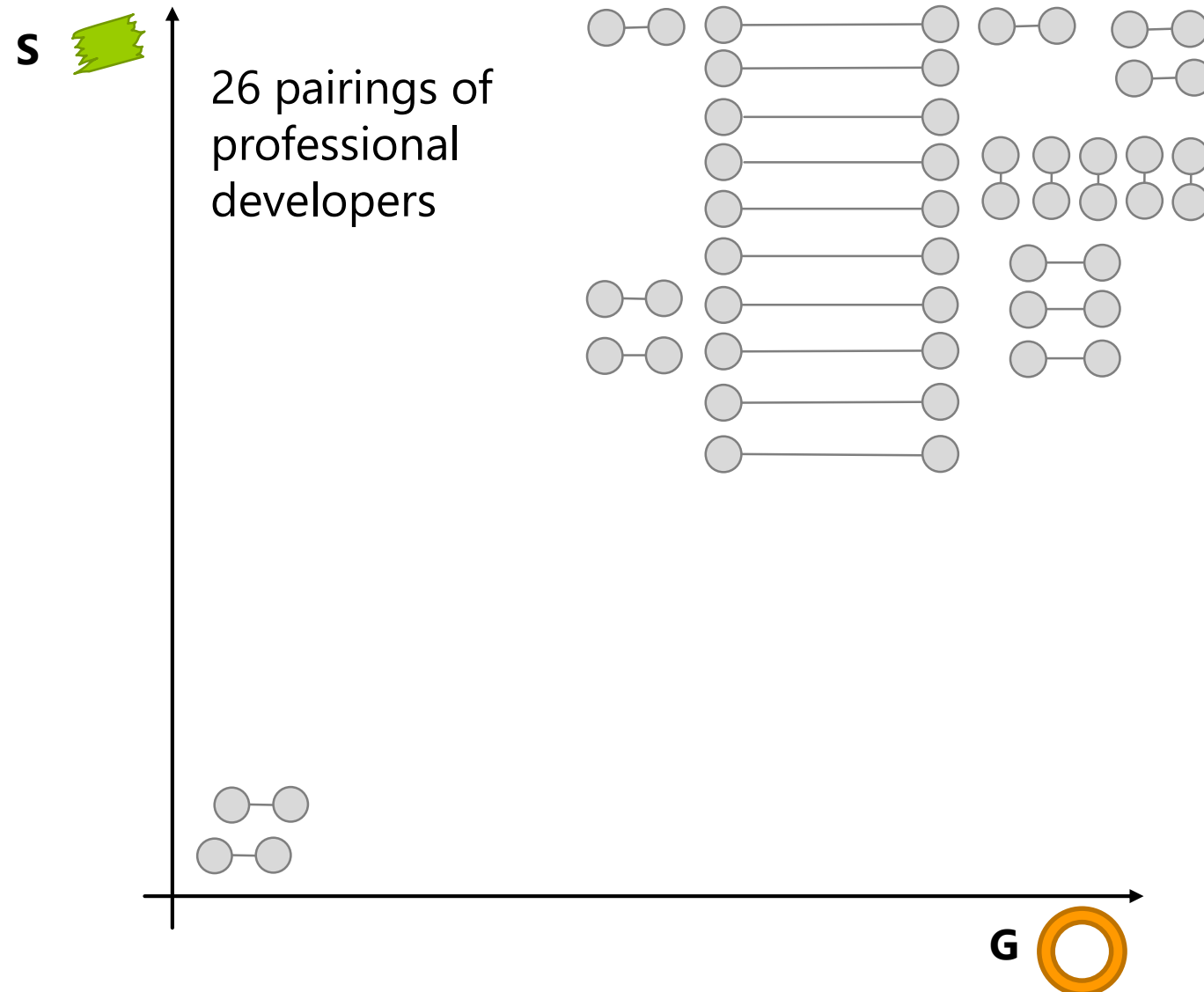
Overall Session Dynamic



Overall Dynamic

1. Close **Primary Gap**
2. Close **Secondary Gap**
3. Seize **G Opportunity**

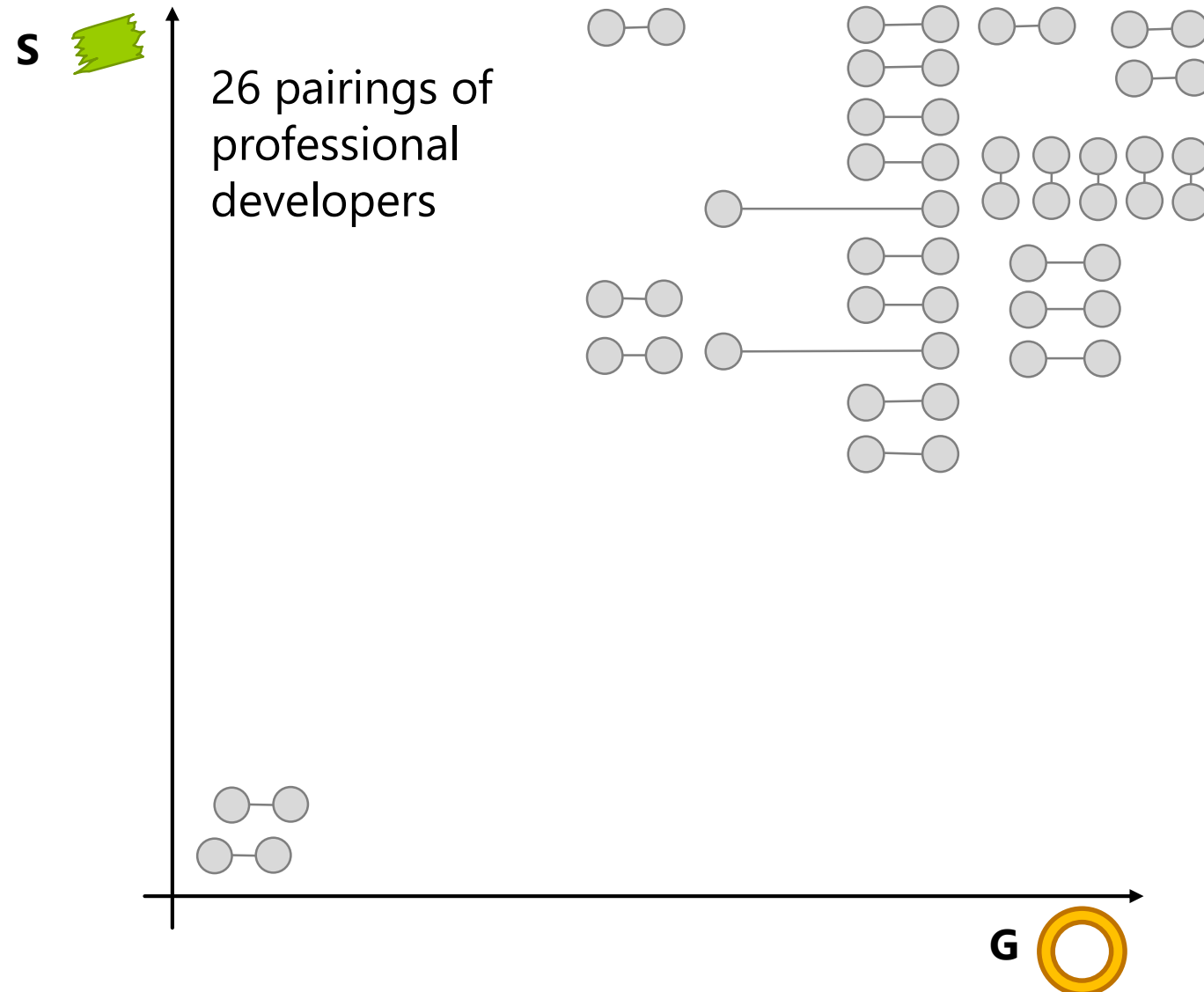
Overall Session Dynamic



Overall Dynamic

1. Close **Primary Gap**
2. Close **Secondary Gap**
3. Seize **G Opportunity**

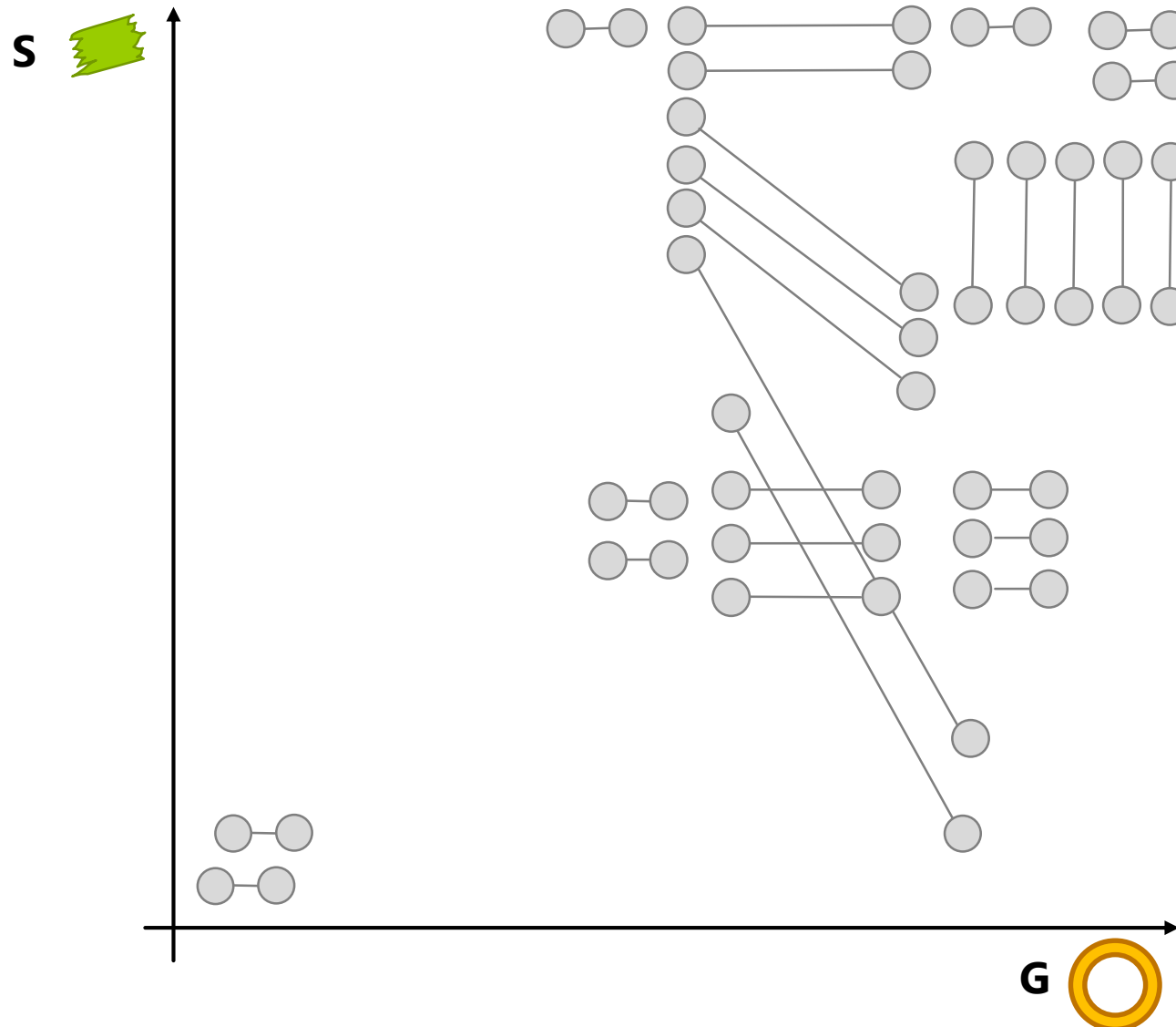
Overall Session Dynamic



Overall Dynamic

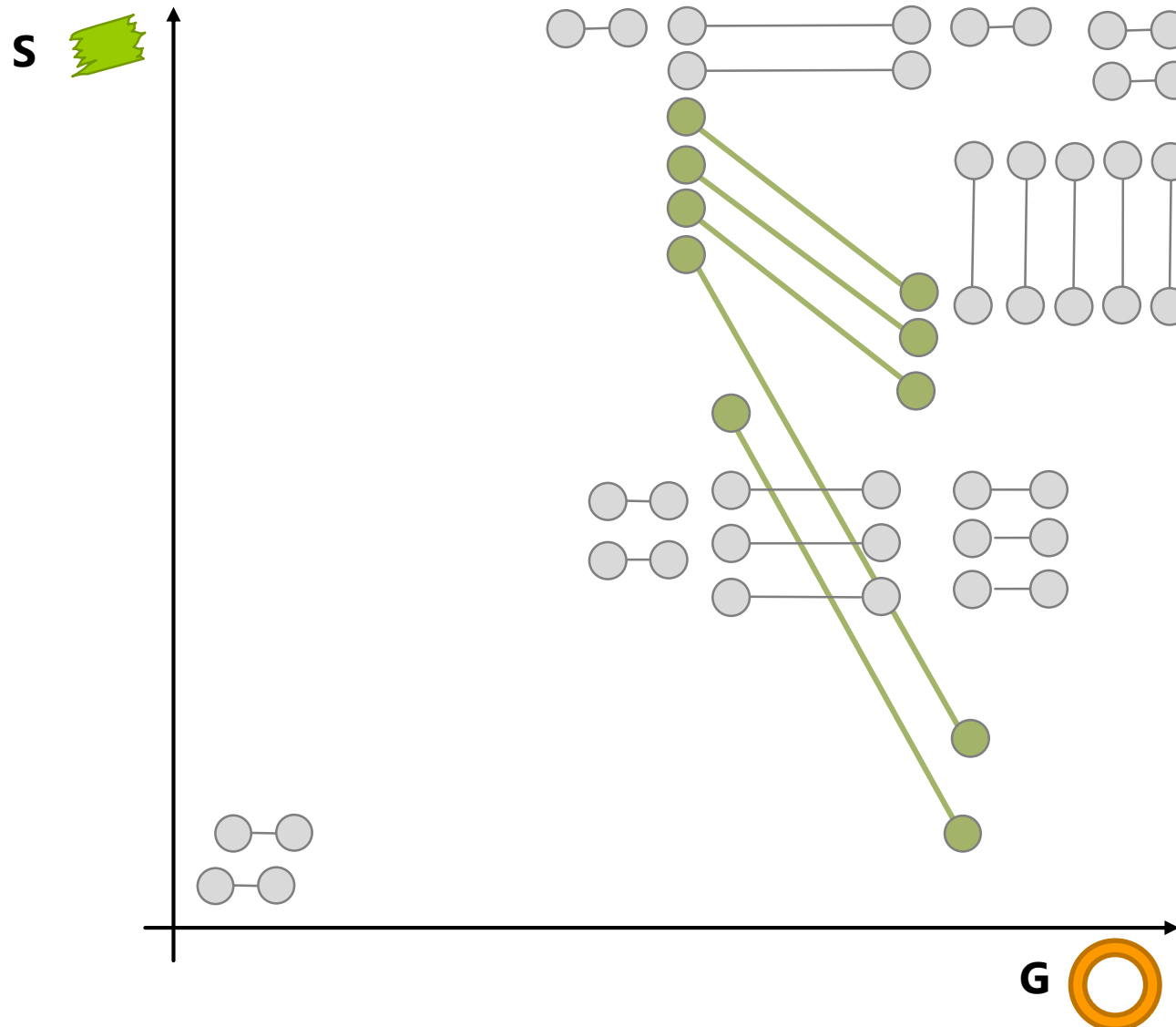
1. Close **Primary Gap**
2. Close **Secondary Gap**
3. Seize **G Opportunity**

Summary



- A lack of and differences in *system understanding* are **more important** than differences in *general programming experience*
- What matters is **task-relevant** knowledge → different knowledge needs, different session dynamic

Summary



- A lack of and differences in *system understanding* are **more important** than differences in *general programming experience*
- What matters is **task-relevant** knowledge → different knowledge needs, different session dynamic
- Mutually satisfactory constellation:

Complementary Gaps

Summary

preprint



<http://inf.fu-berlin.de/inst/ag-se/pubs/ZiePre20-ppsessiondyn.pdf>

ICSE '20, May 23–29, 2020, Seoul, Republic of Korea

Considering the session EA1 as a whole, the long-running *push* episode in the beginning enabled the pair to work as peers to then solve the actual task together. In our data, this is common behavior in the beginning of a session, especially when one developer already worked on the task before. It occurred to us that these pairs start with an *asymmetric* situation regarding their *S need* and they turn it into a *symmetric* one which allows them to make further progress closely together (*co-produce* episodes in the case of EA1). So achieving *S need* symmetry comes first (closing the *primary gap*), acquiring the remaining *S knowledge* together follows (closing the *secondary gap*).

5.4 Session Visualizations

Below, we will provide schematic representations of pairs' knowledge constellations and their trajectory throughout several example sessions (Fig. 2). Each developer is represented by a point on a two-dimensional coordinate system, with the degree of *G need* decreasing from left to right and the degree of *S need* from bottom to top. In a *qualitative* sense, the vertical distance between a pair's two points hence represents the *primary gap*, the distance from the top represents the *secondary gap*, and the partner's horizontal distance represents the *G opportunity*.

The pair's points are drawn at their *initial constellation*. The reduction of knowledge gaps is indicated by arrows originating at the developer whose understanding improves: upward for increasing *S knowledge*, to the right for improved *G knowledge*. Multiple arrows starting at the same height indicate multiple attempts made to address a *Knowledge Need*. The trajectories do not depict technical progress at all. Arrow length does not represent time at all, but only the (qualitative) reduction of a knowledge gap. Arrow color indicates the *mode* in which the knowledge gap is narrowed (see Section 2.2 and [33] for details). The numbers in the trajectories correspond to numbers in the article text. For readability, a single arrow (e.g., \rightarrow) might represent multiple knowledge transfer episodes pertaining to similar topics.

6 SESSION DYNAMICS PROTOTYPES

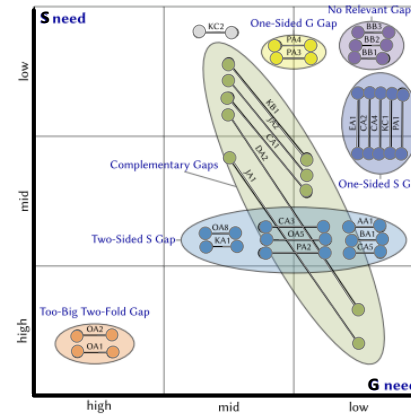
We will now describe how pairs actually deal with their *Knowledge Needs*: We have identified six different *initial constellations* in our data, each having a different combination of *primary* and *secondary gaps* and *G opportunity*, each leading to a characteristic session dynamic. These form a set of session dynamics *prototypes* that is very useful to understand how PP works and when (and for which goals) it is most useful. See Fig. 1 for the *initial constellations* of all analyzed sessions. Other *initial constellations* are conceivable, but we have not seen them.

6.1 No Knowledge Gaps, No Opportunity

In many pair programming sessions, there is a point at which both partners have all necessary system understanding as well as all needed general programming knowledge to work productively on the task. This is the *No Relevant Gaps* constellation (see Fig. 1). The only pair we have seen that had this as its *initial constellation* forms our first and simplest example.

6.1.1 Example 1: Greenfield Development. Developers B1 and B2 work on a new feature from scratch over the course of one afternoon

Franz Zieris and Lutz Prechelt



S and *G need* denote a developer's gap in task-relevant knowledge with regard to the specific software system and generic software development, respectively (see Section 4.1). Each pair of points represents one PP session (see Table 1); sessions are grouped in six recurring pair constellations.

Figure 1: Initial pair constellations of the analyzed sessions

in three sessions (BB1 to BB3) with short pauses in between. Both are proficient in the involved technologies and need to interact with existing code only through few and well-understood interfaces (neither *S* nor *G need*). The newly developed code stays small enough to be fully understood by both developers at all times. Apart from a short orientation phase in the beginning, when they decide on where to visually place the feature in the GUI, they are in construction-only mode throughout the session, i.e., defining requirements and discussing design proposals, with zero debugging. B1 and B2 develop no *S need* through all three sessions while the body of *S knowledge* increases along the way.

6.1.2 Discussion. This nice and easy *initial constellation* is likely only in development-from-scratch situations, which are not frequent. It is fragile and can be destroyed by any lengthy debugging episode or by between-session pauses long enough to let developers forget relevant details of what they built.

6.2 Dealing with a One-Sided *S Gap*

In some pair programming sessions, one developer has an *S advantage*, e.g., because she already started work on the task. Two constellations have this property: *One-Sided S Gap* and *Complementary Gaps*, each of which happened to be the initial constellation of five of our analyzed sessions (see Fig. 1). Whenever a one-sided *S gap*—the *primary gap*—exists, the pair addresses it first.

In most cases, the developer with the larger *S need* is aware of the gap and the pair can address it proactively, as illustrated in Section 6.2.1. If, however, she is not aware of her *S need*, she

- A lack of and differences in *system understanding* are **more important** than differences in *general programming experience*

- What matters is **task-relevant** knowledge → different knowledge needs, different session dynamic

- Mutually satisfactory constellation: **Complementary Gaps**

Images



<https://web.archive.org/web/20080509191418/http://www.cenqua.com/pairon/>



Icon "design" by Adrien Coquet from the Noun Project



Icon "Bug" by Minh Do from the Noun Project



Icon "knowledge" by Olivia from the Noun Project



Icon "Box" by No More Heroes from the Noun Project



Icon "corner webs" by Kate Maldjian from the Noun Project



Icon "Computer" by Denis Shumaylov from the Noun Project