

Getting Feedback in 100PM

Stephan Brandauer

Part 1

What is "100PM"?

100PM

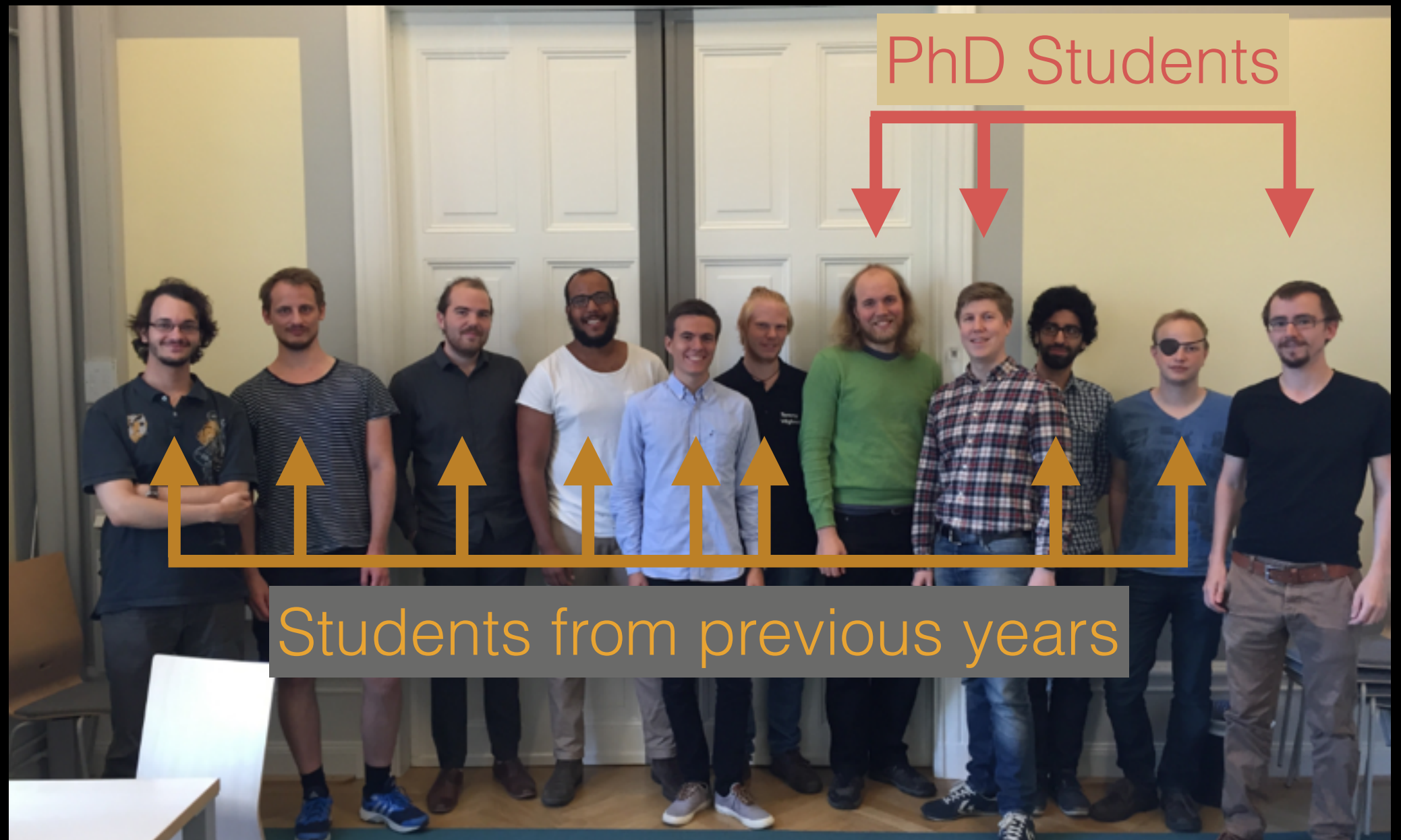
- “Imperative and Object-Oriented Programming Methodology”
- 2nd Year Programming Course
- Real Scary Stuff (20 Credits!)



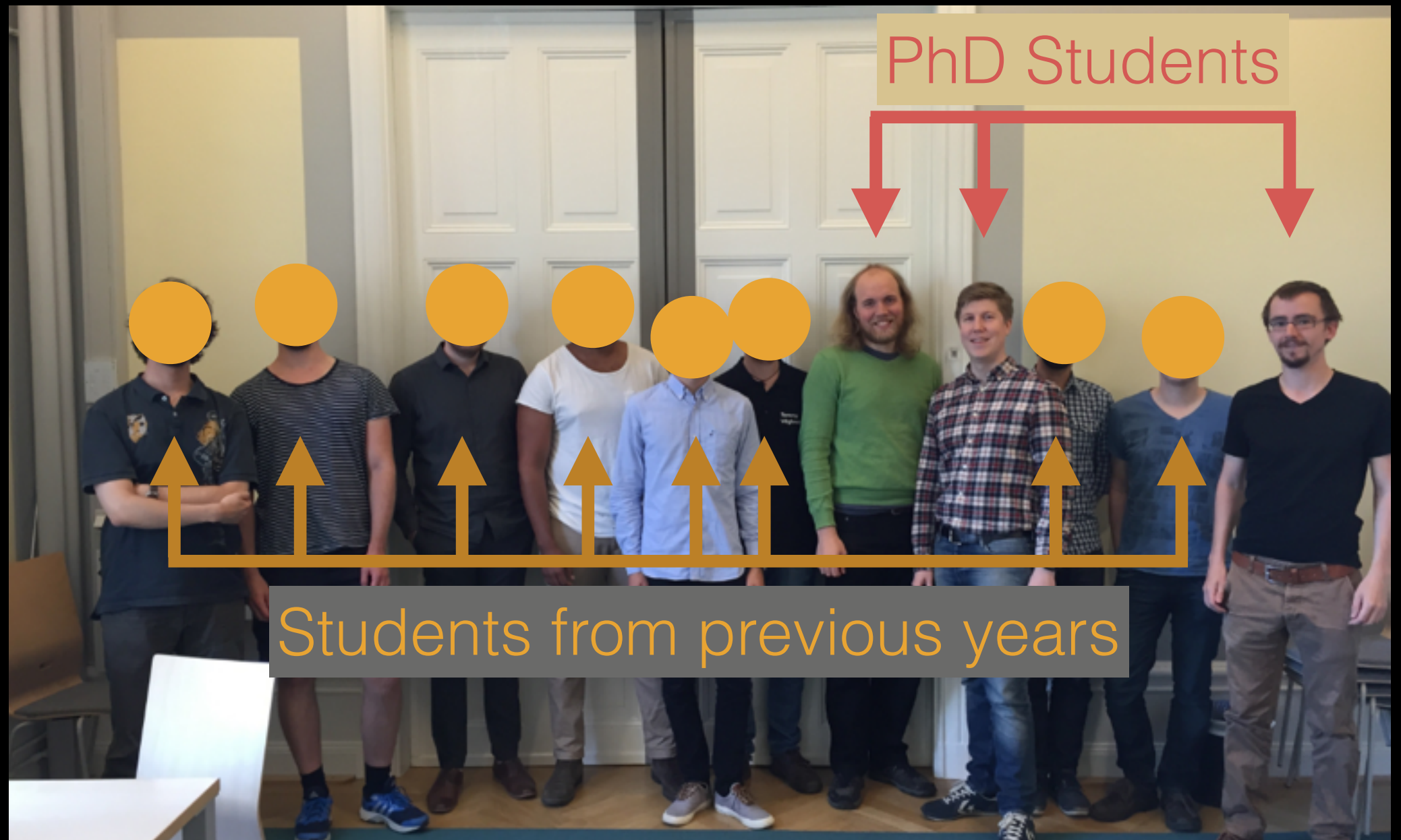
Lots of TAs!
(~12 TAs in 2015)



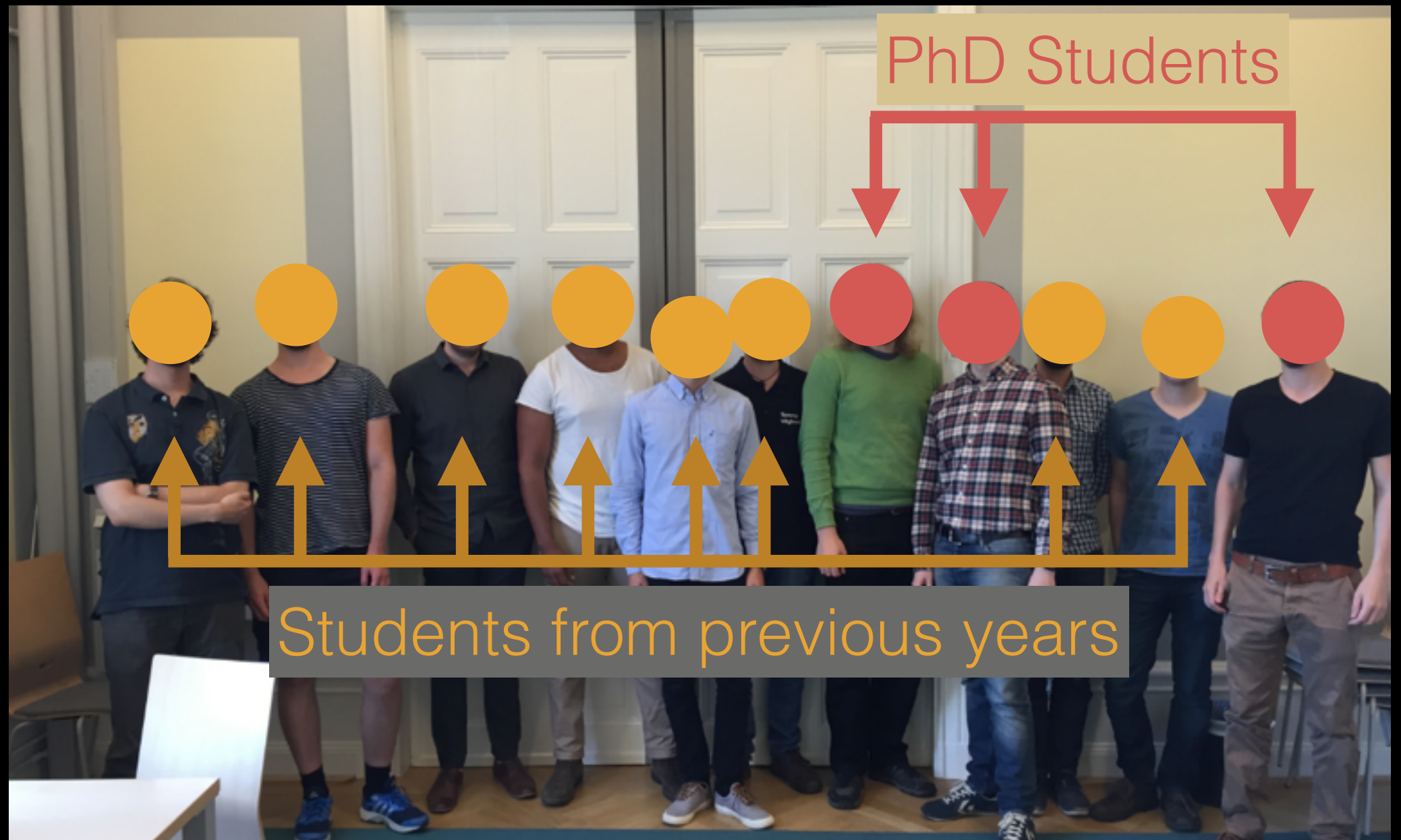
Lots of TAs!
(~12 TAs in 2015)



Lots of TAs!
(~12 TAs in 2015)



Lots of TAs!
(~12 TAs in 2015)



Lots of TAs!
(~12 TAs in 2015)

Even More Students!

145 students

Even More Students!



145 students

Even More Students!



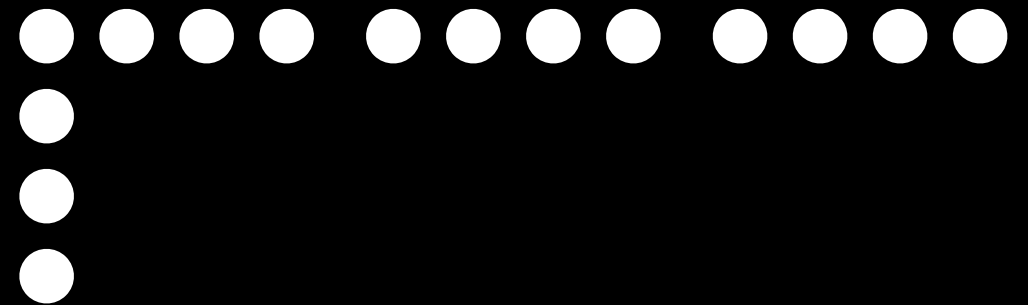
145 students

Even More Students!



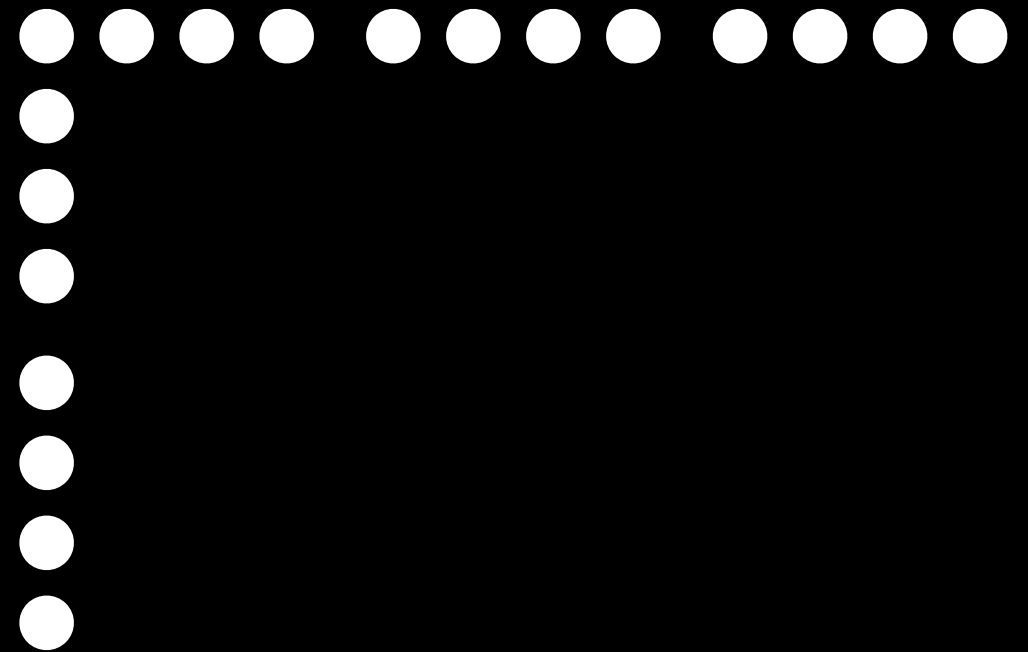
145 students

Even More Students!



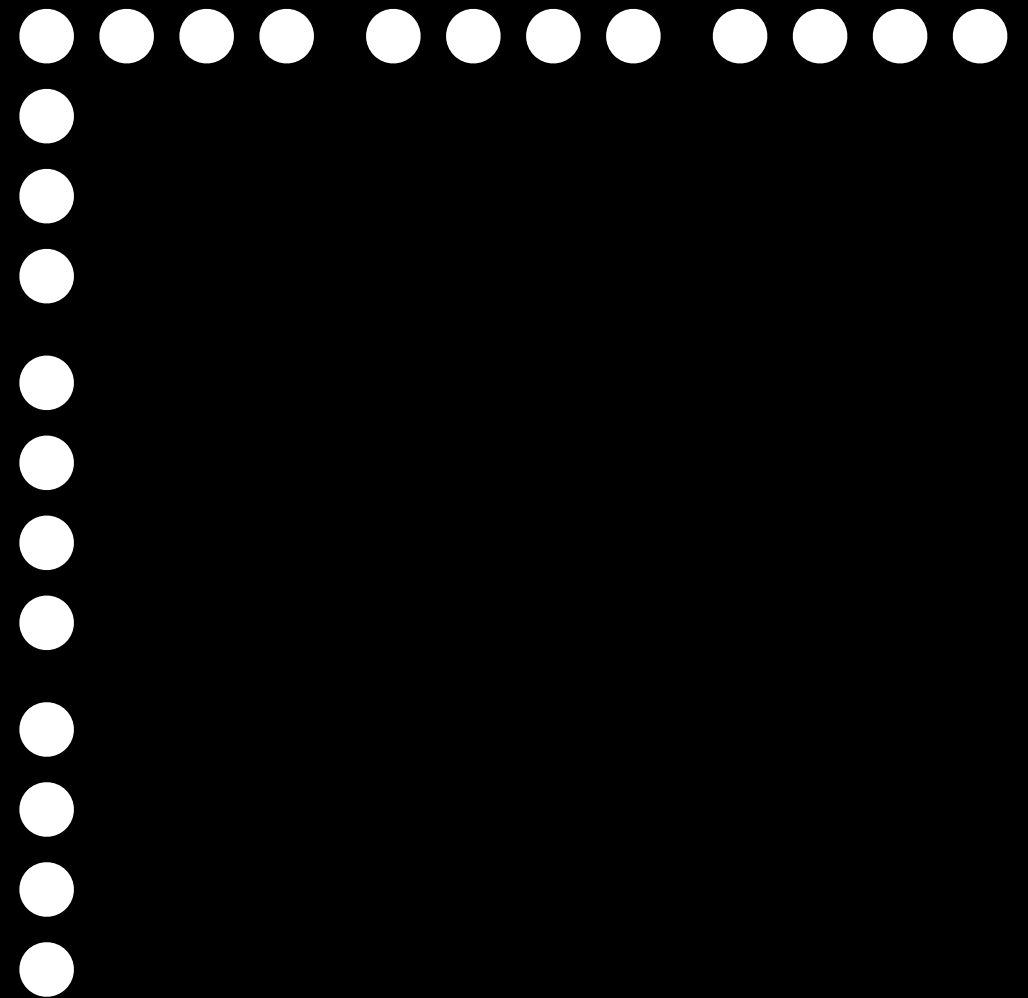
145 students

Even More Students!



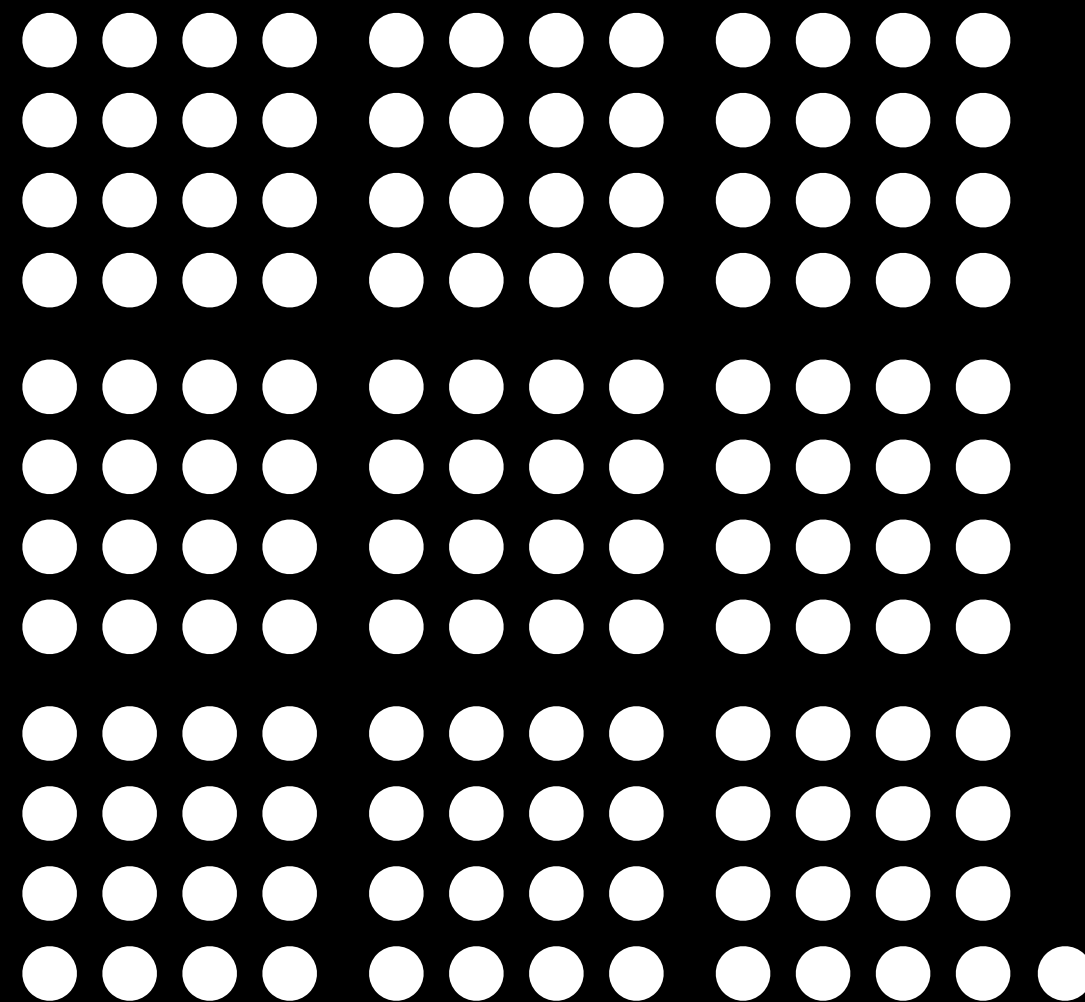
145 students

Even More Students!



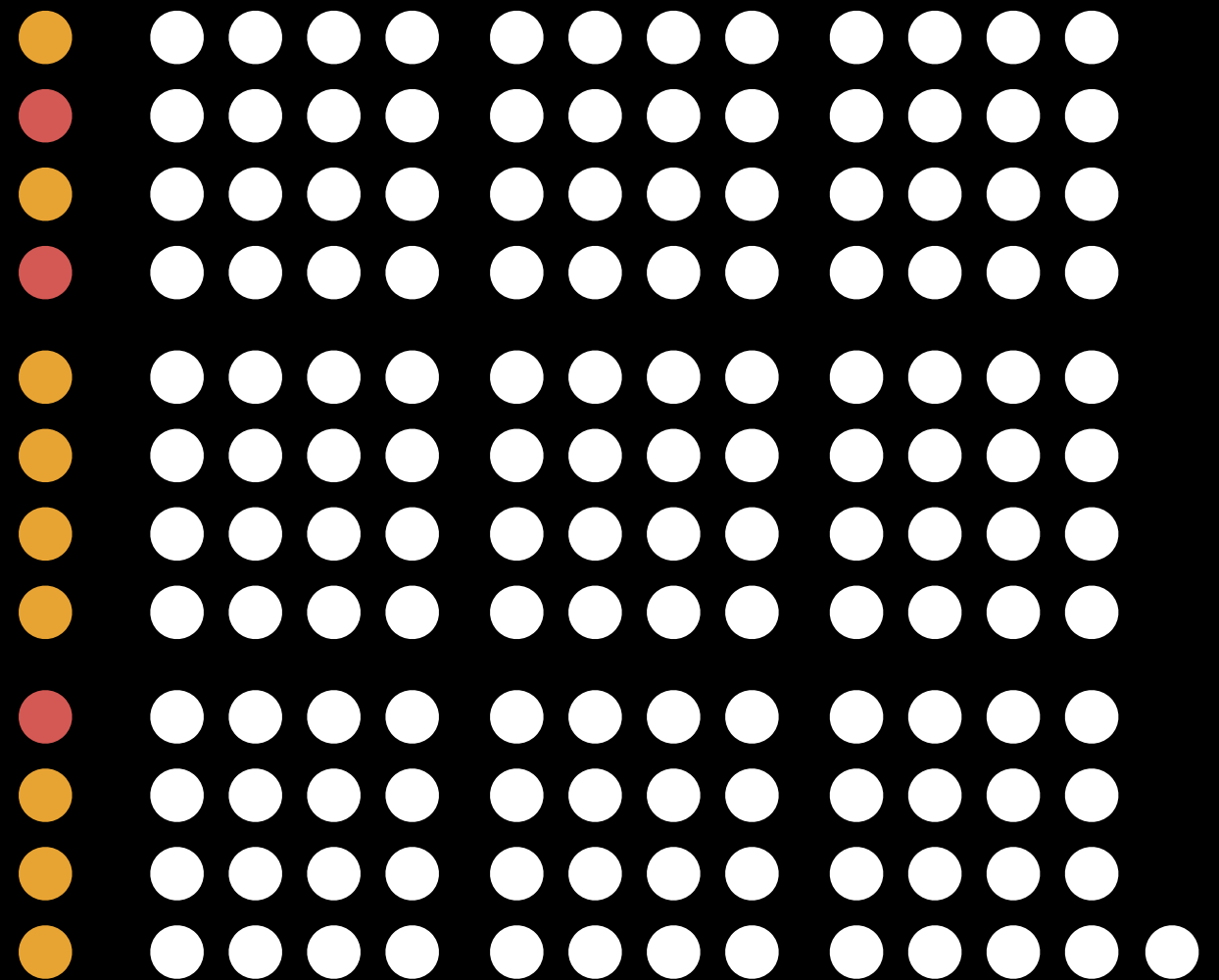
145 students

Even More Students!



145 students

Even More Students!

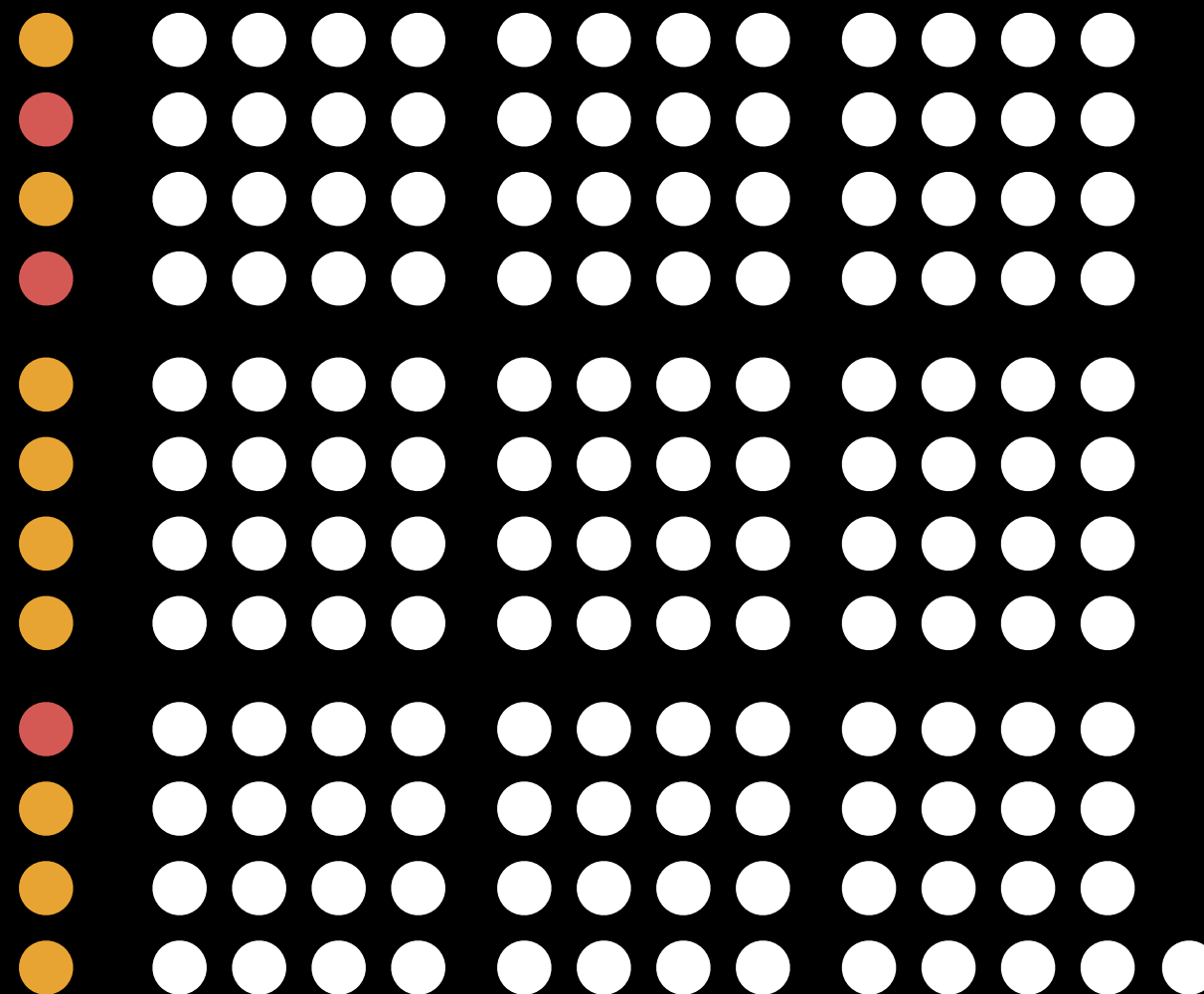


145 students

Even More Students!

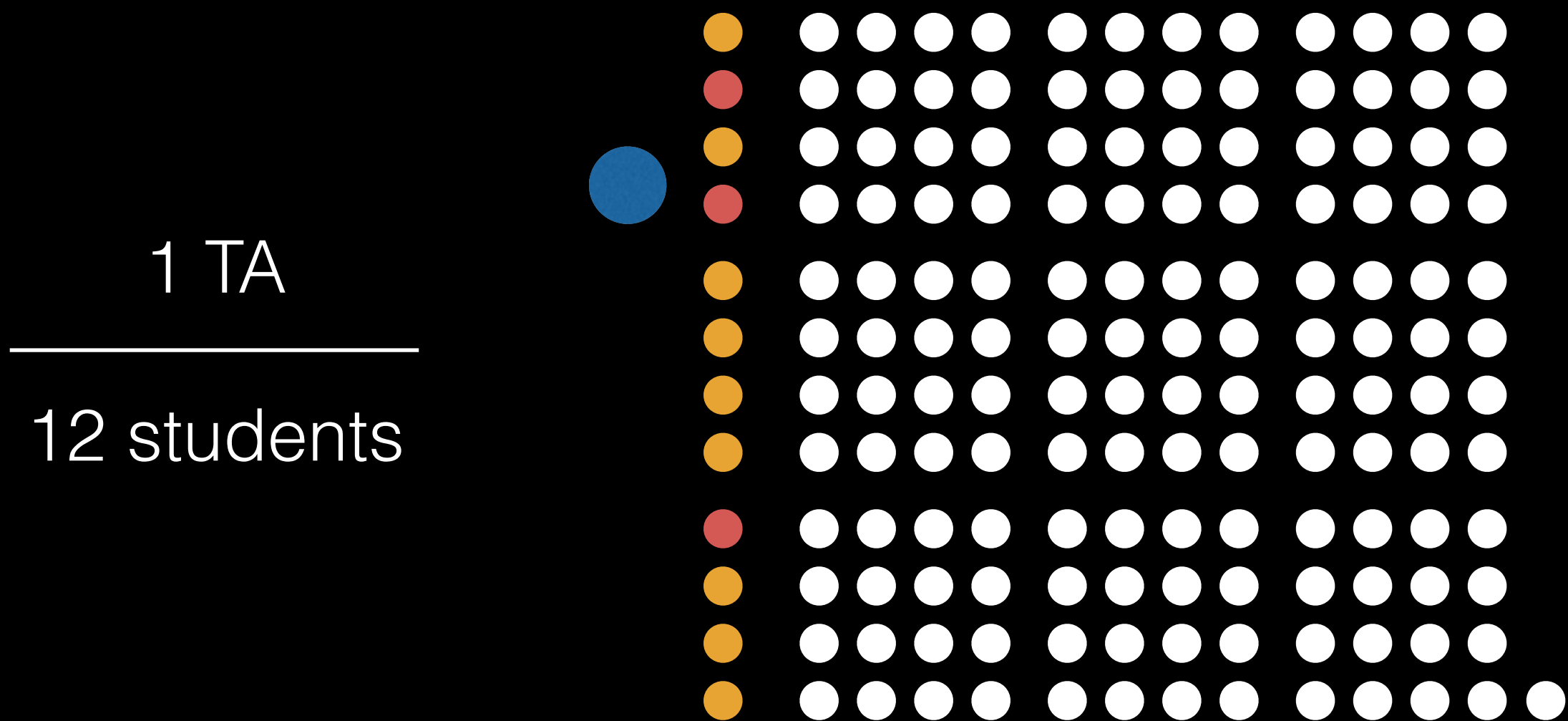
1 TA

12 students



145 students

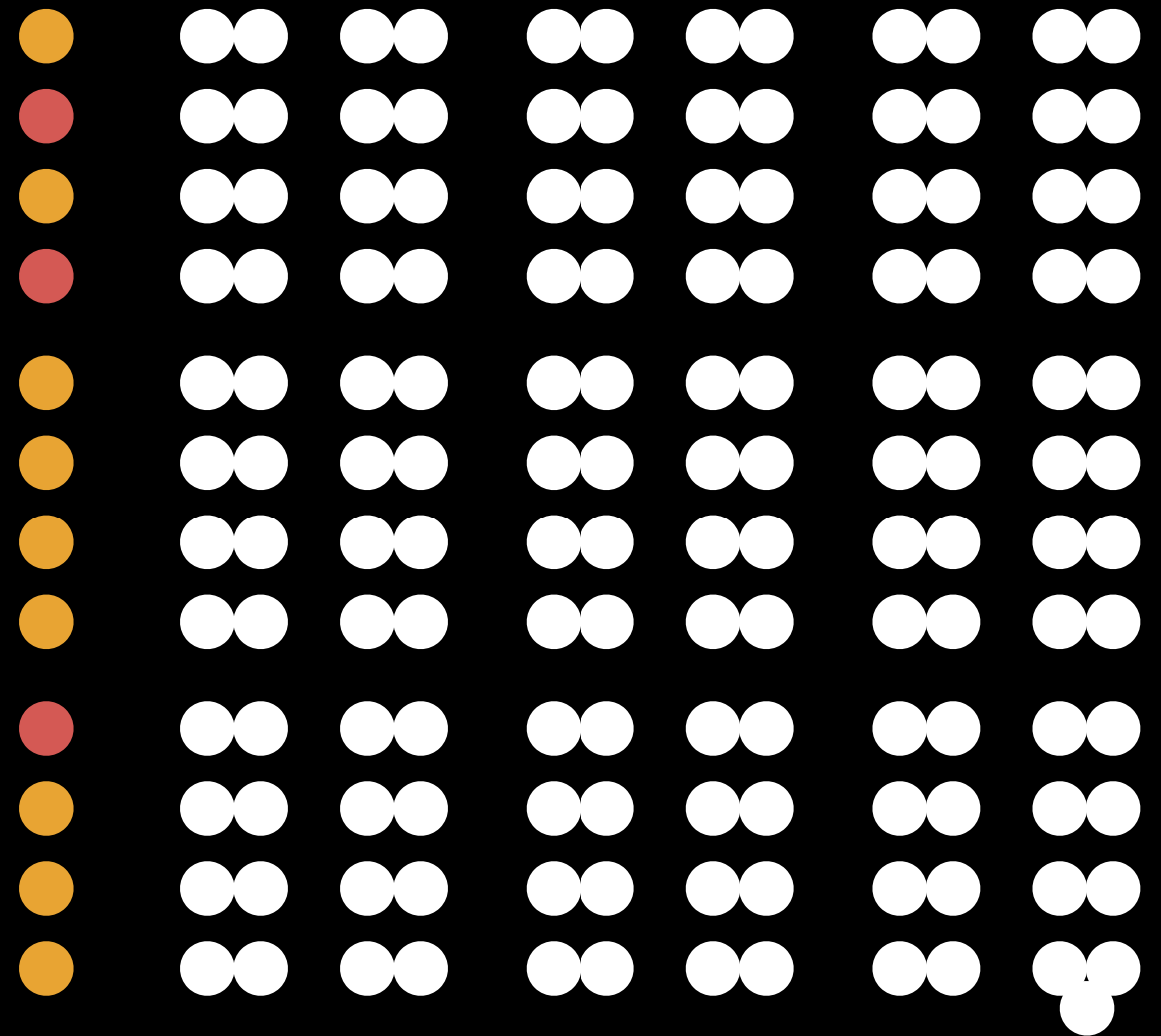
Even More Students!



145 students

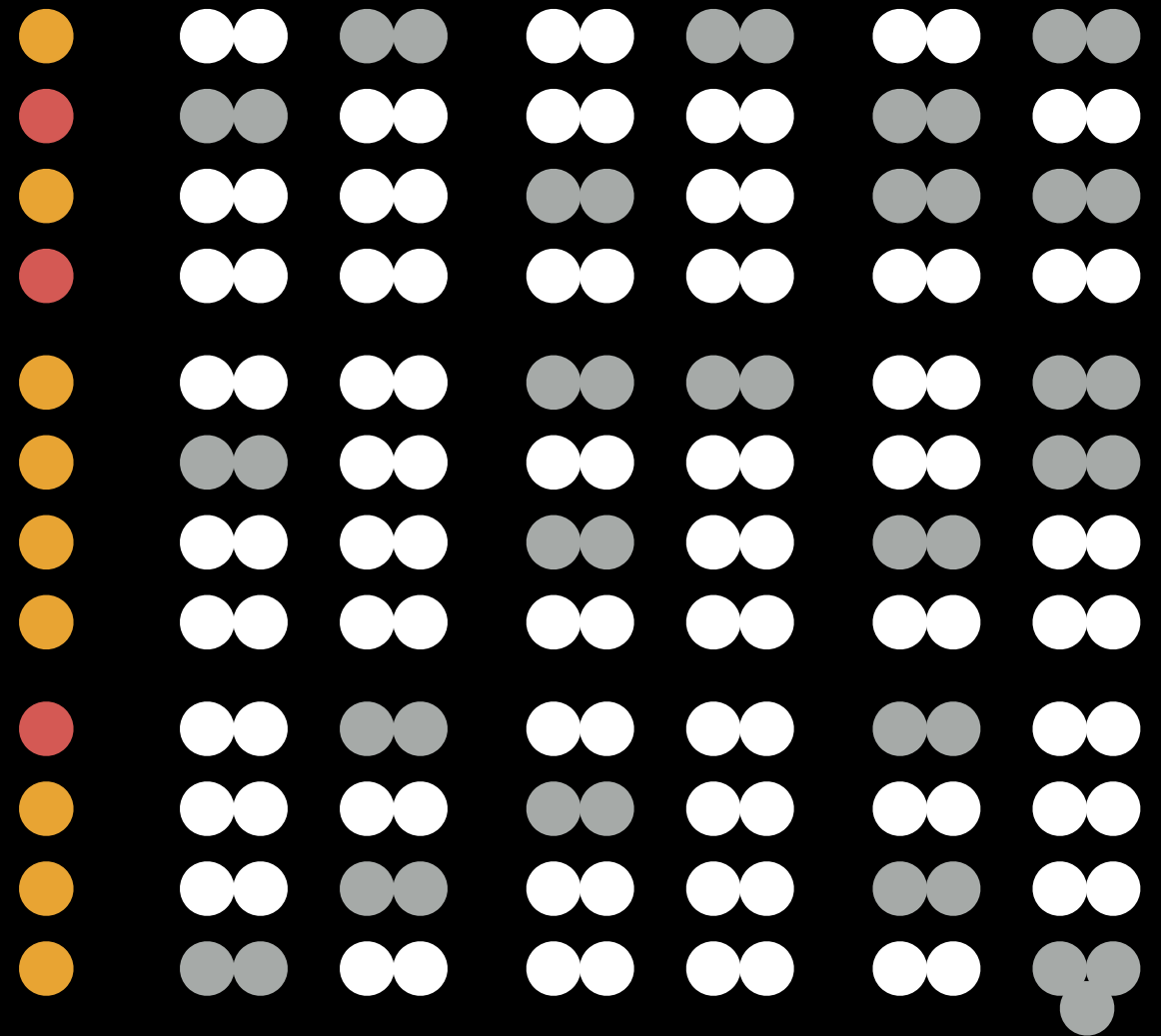
Phase 1: C and Java

All start with
same exercise



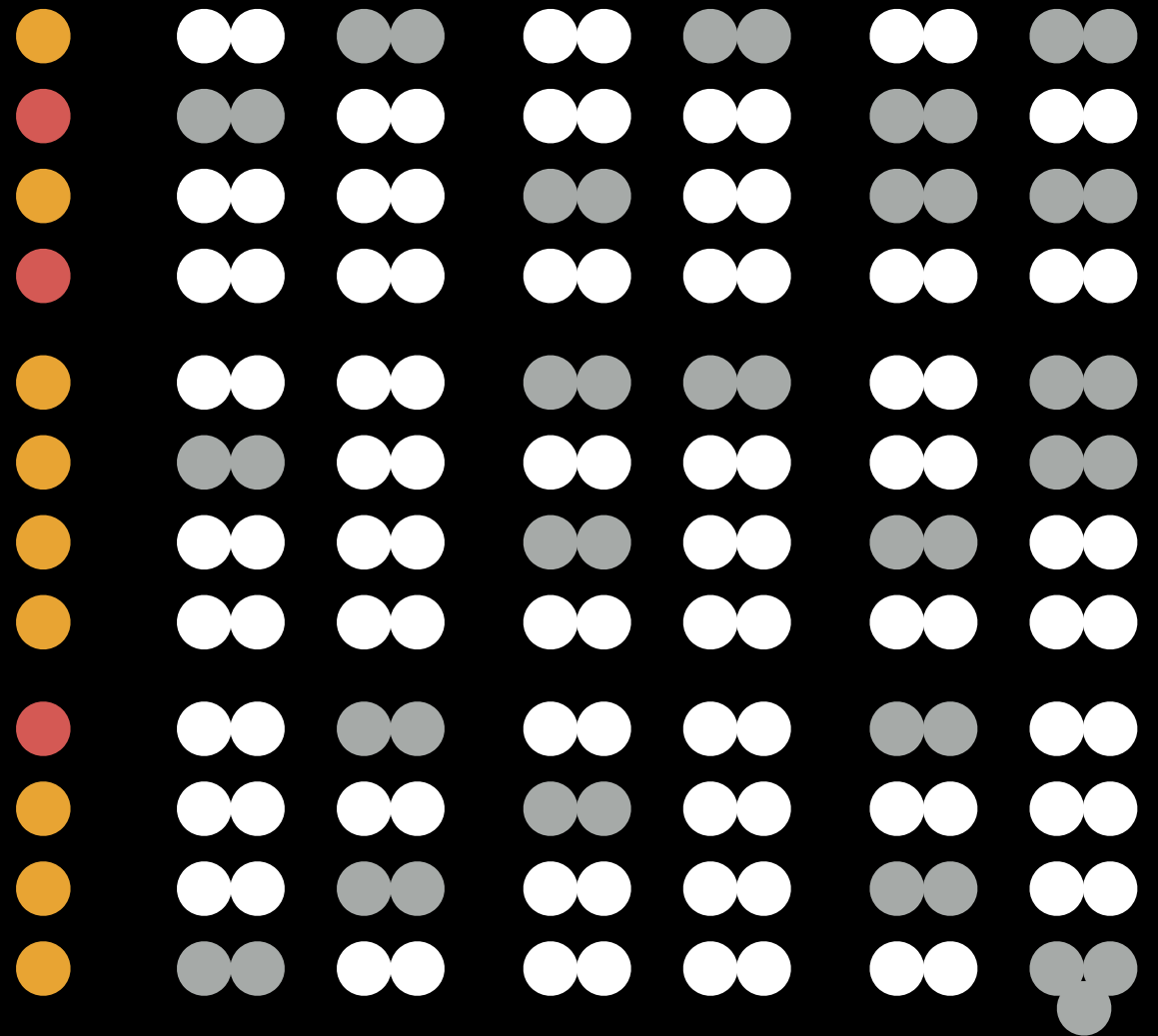
Phase 1: C and Java

Pairs continue
with exercise,
or pick another one



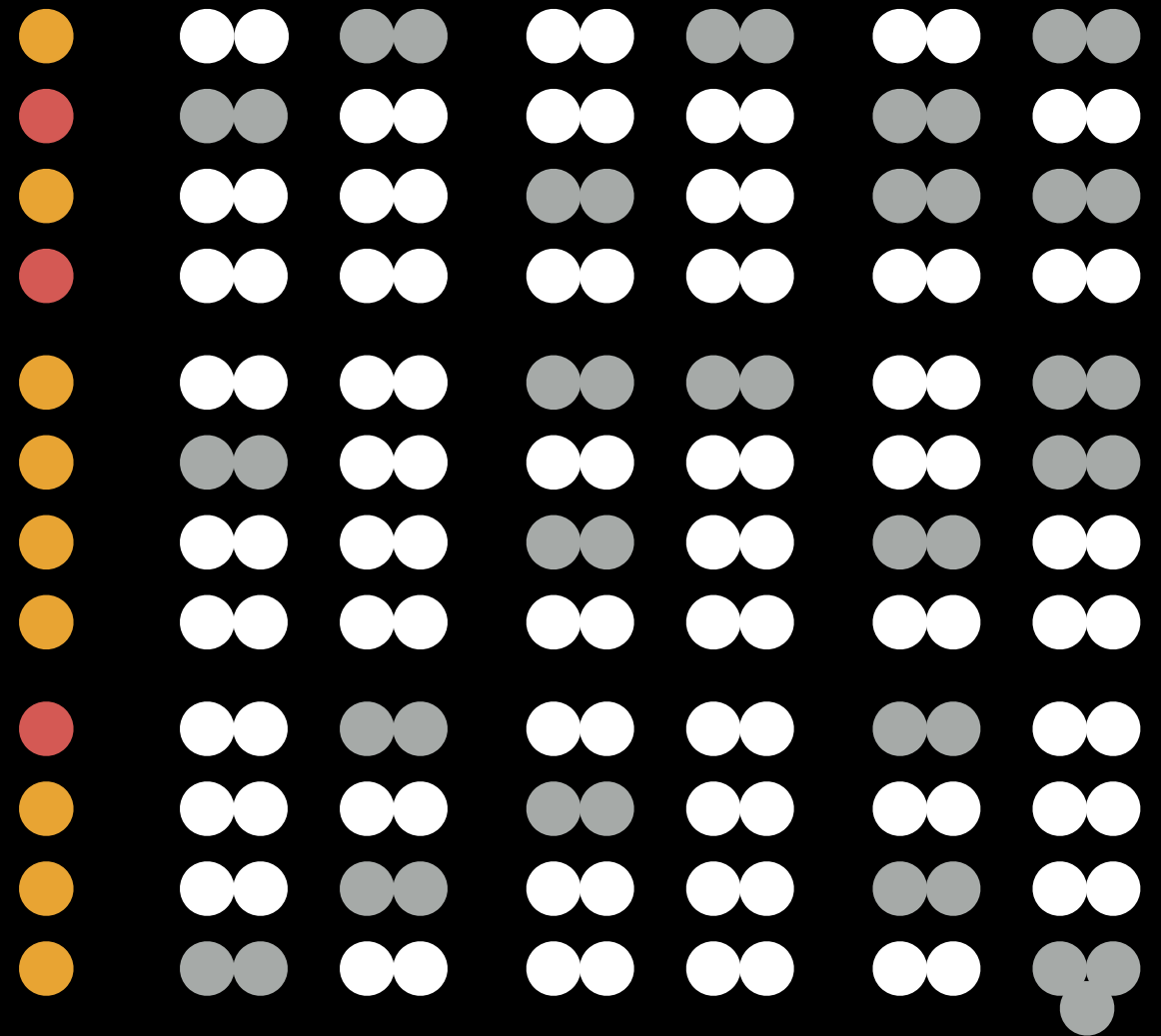
Phase 1: C and Java

Pairs can change,
we have little
hand in this.



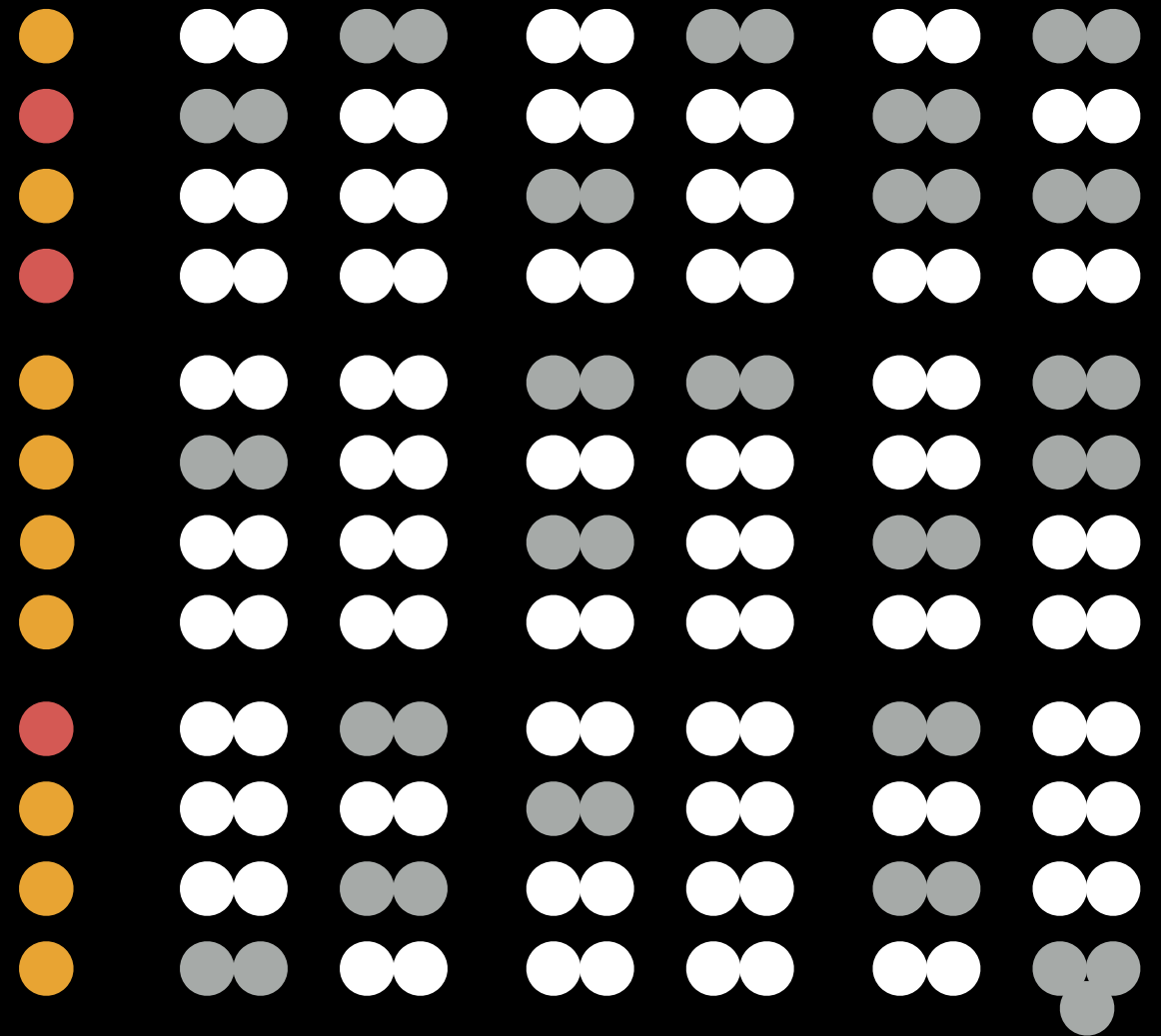
Phase 1: C and Java

Pairs can change,
we have little
hand in this.



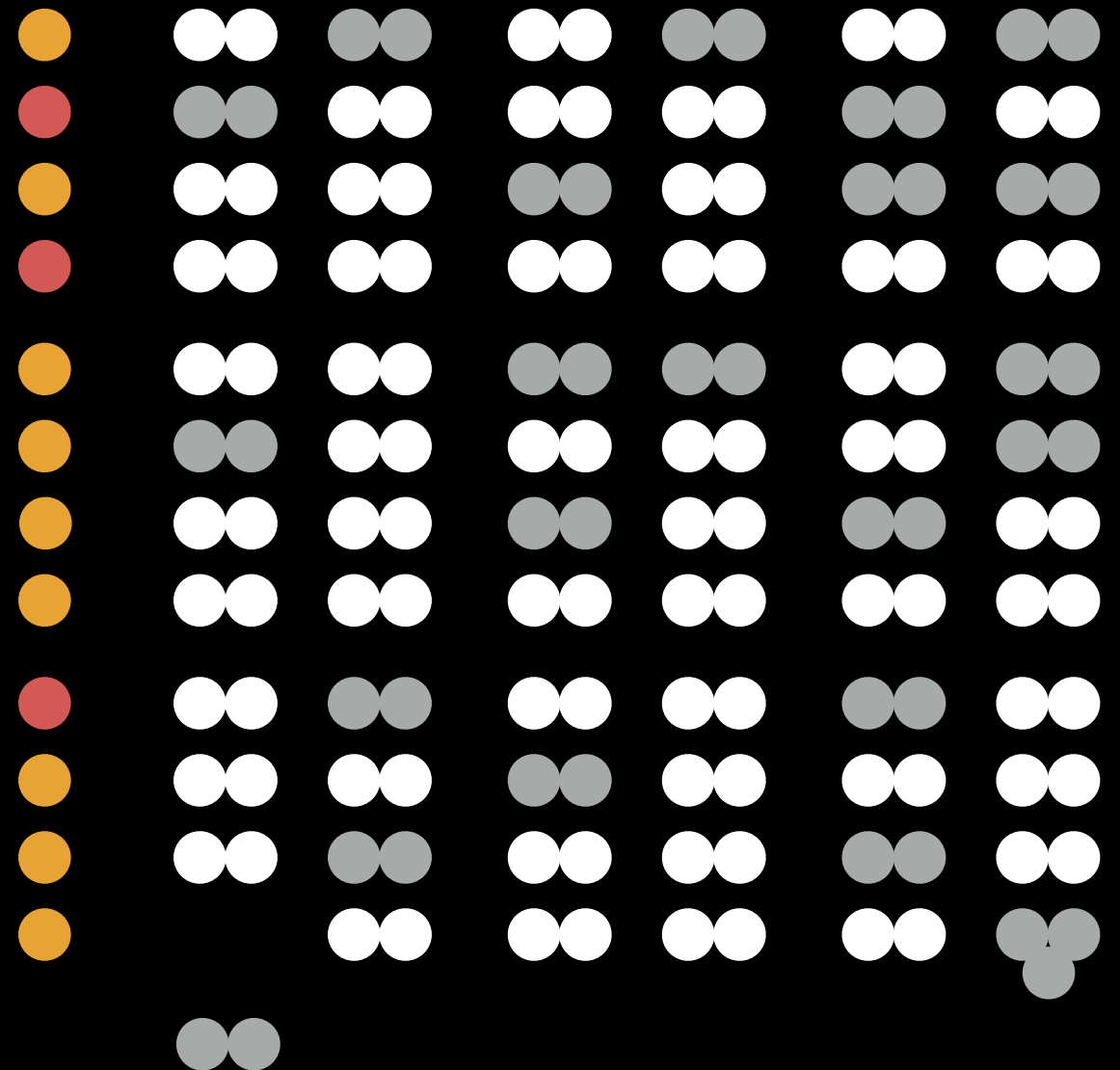
Phase 1: C and Java

Pairs demonstrate
learning progress
to TAs — in the lab.
Oral examination.



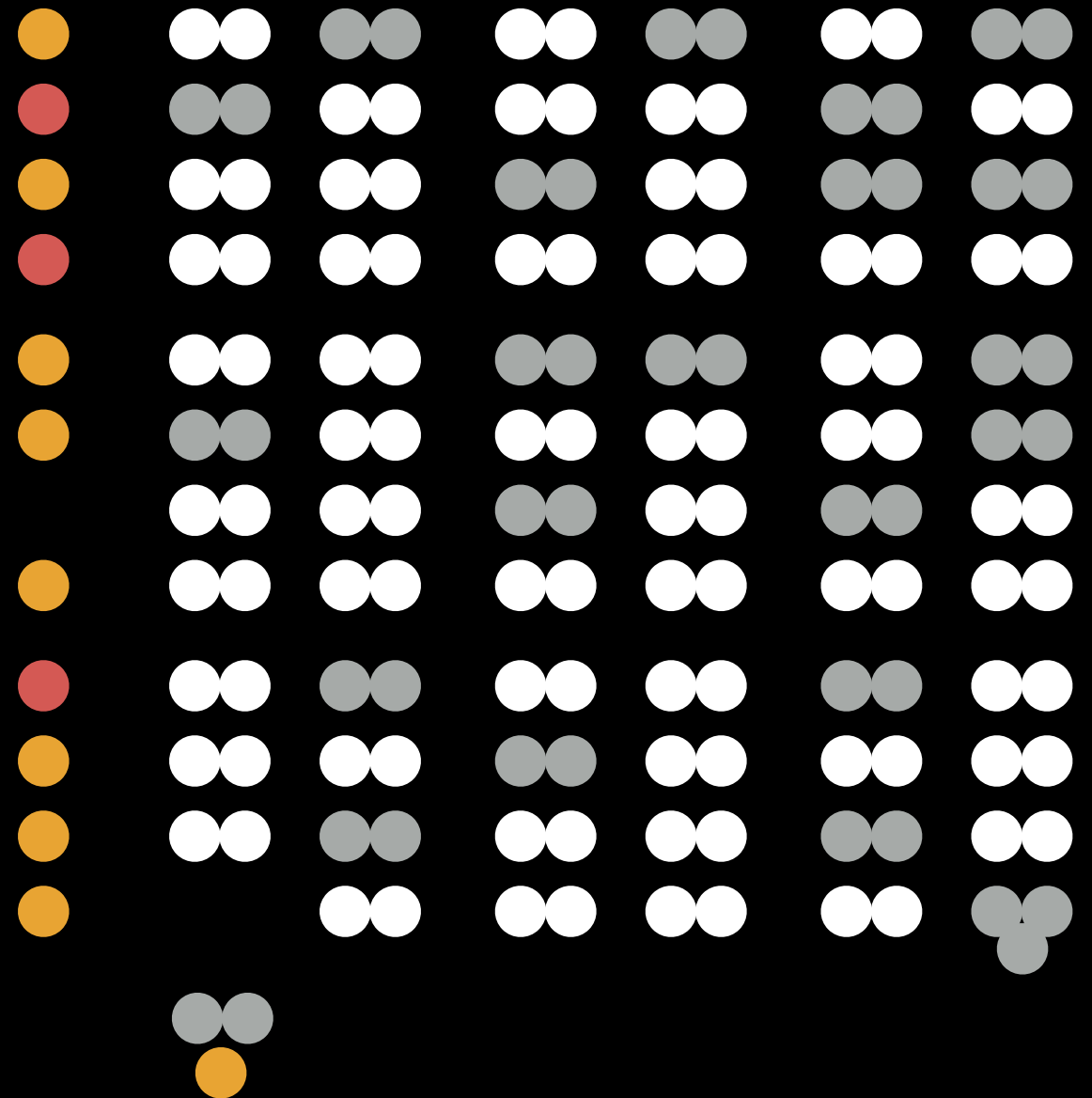
Phase 1: C and Java

Pairs demonstrate
learning progress
to TAs — in the lab.
Oral examination.



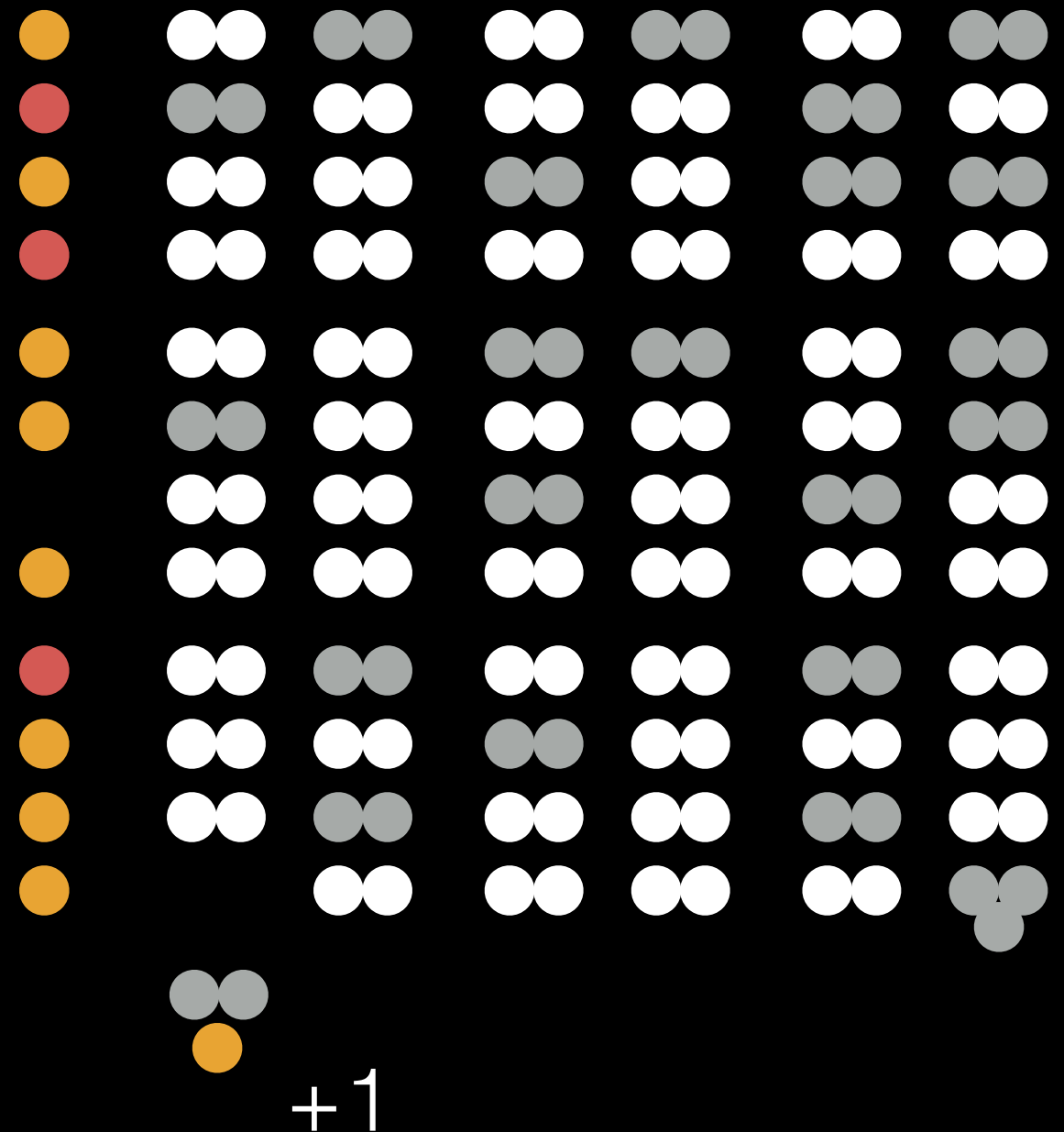
Phase 1: C and Java

Pairs demonstrate
learning progress
to TAs — in the lab.
Oral examination.



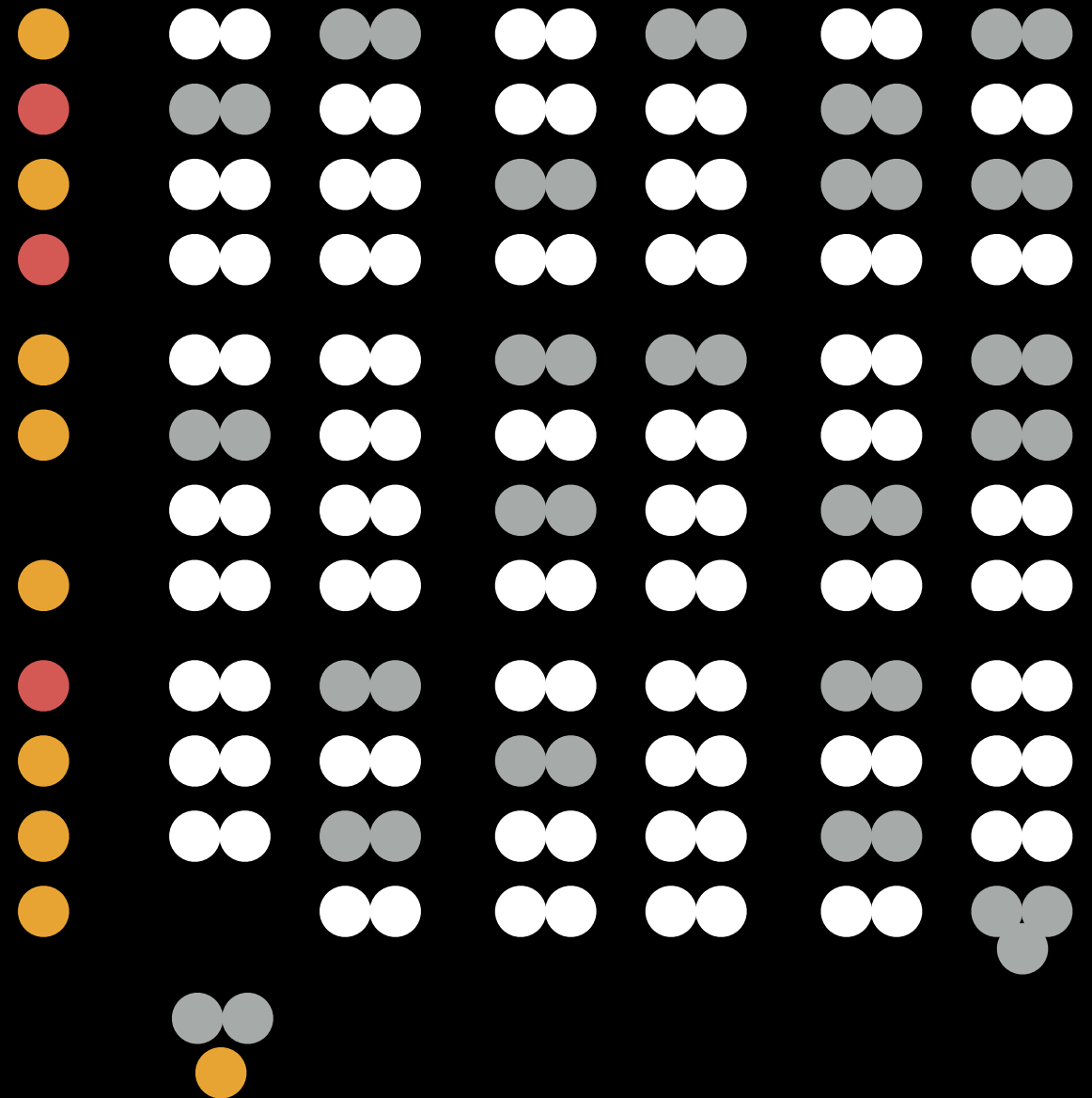
Phase 1: C and Java

Pairs demonstrate
learning progress
to TAs — in the lab.
Oral examination.



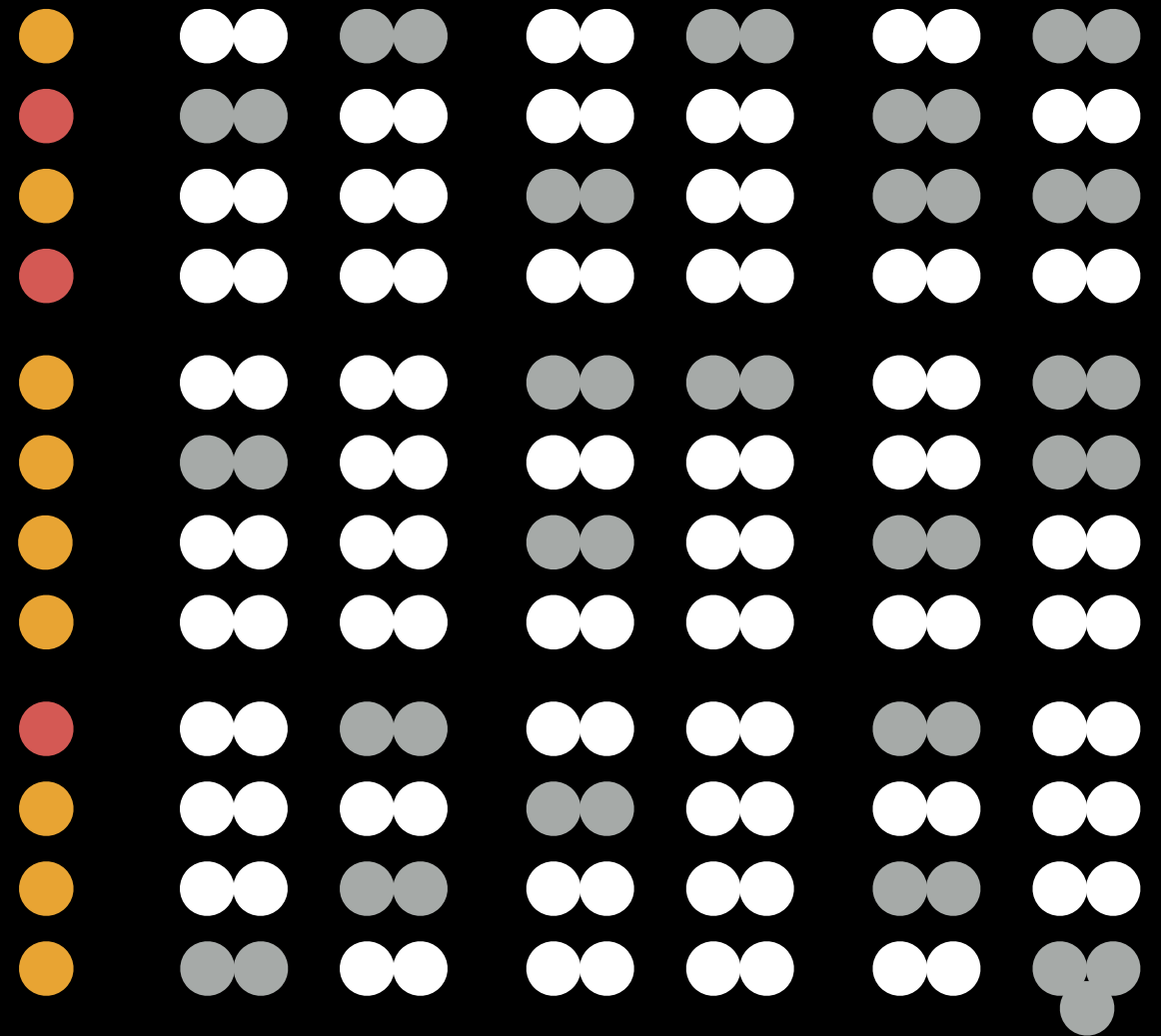
Phase 1: C and Java

Pairs demonstrate
learning progress
to TAs — in the lab.
Oral examination.



Phase 1: C and Java

Pairs demonstrate
learning progress
to TAs — in the lab.
Oral examination.



Phase 1: C and Java



Home Achievements Sign in

Unlockable Achievements (aka kursmål/kunskapsmål)

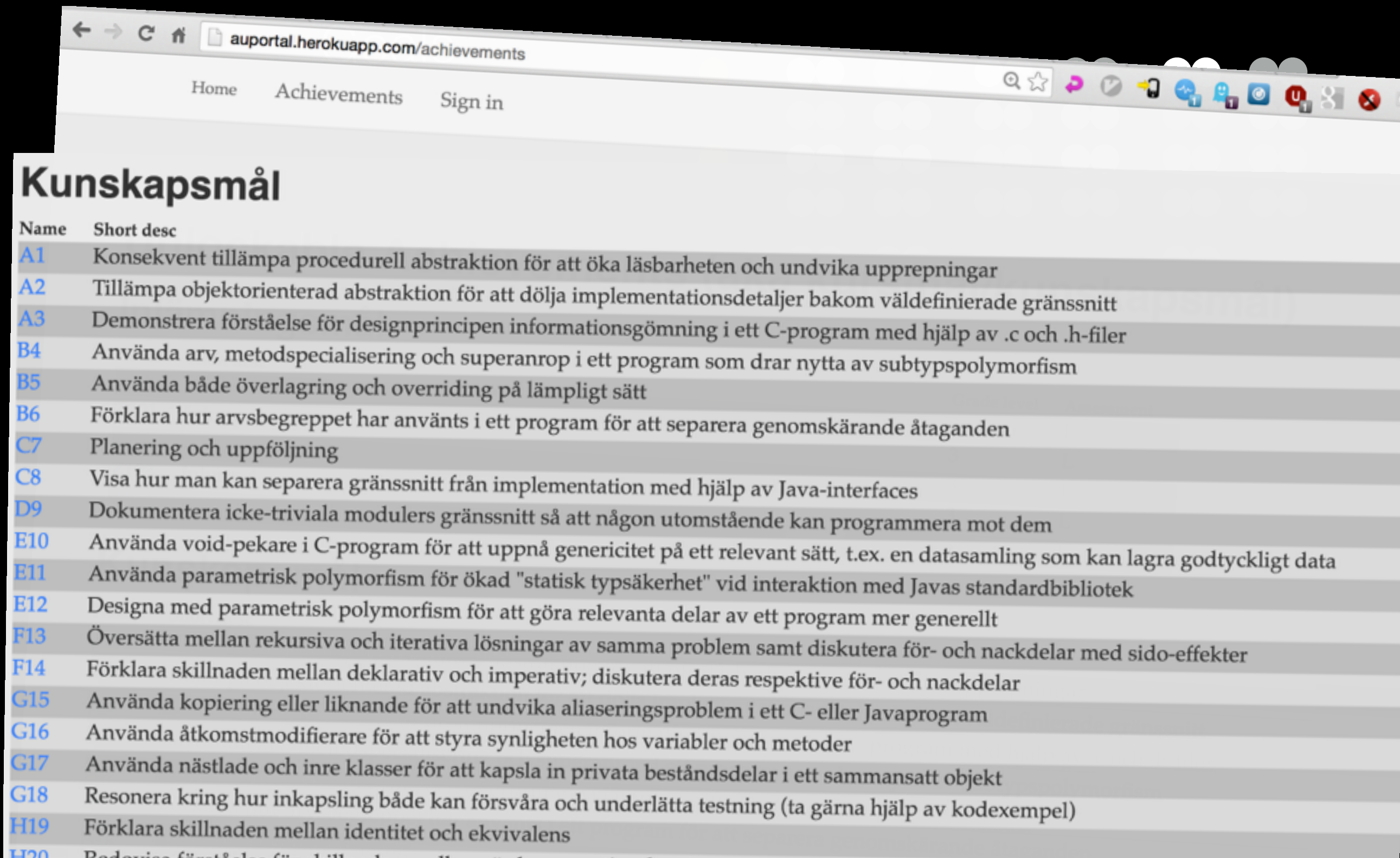
Inlämningsmål

Name	Short desc	Grade level	Assessment
Z100	Inlupp 1	3	L
Z101	Inlupp 2	3	L
Z102	Inlupp 3	3	L
Z103	Inlupp 4	3	L
Z104	Inlupp 5	3	L

Kunskapsmål

Name	Short desc
A1	Konsekvent tillämpa procedurell abstraktion för att öka läsbarheten och undvika upprepningar
A2	Tillämpa objektorienterad abstraktion för att dölja implementationsdetaljer bakom väldefinierade gränssnitt
A3	Demonstrera förståelse för designprincipen informationsgömning i ett C-program med hjälp av .c och .h-filer
B4	Använda arv, metodspecialisering och superanrop i ett program som drar nytta av subtypspolymorfism
B5	Använda både överlagring och overriding på lämpligt sätt
B6	Förklara hur arvsbegreppet har använts i ett program för att separera genomskärande åtaganden
C7	Planering och uppföljning
C8	

Phase 1: C and Java



Name	Short desc
A1	Konsekvent tillämpa procedurell abstraktion för att öka läsbarheten och undvika upprepningar
A2	Tillämpa objektorienterad abstraktion för att dölja implementationsdetaljer bakom väldefinierade gränssnitt
A3	Demonstrera förståelse för designprincipen informationsgömning i ett C-program med hjälp av .c och .h-filer
B4	Använda arv, methodspecialisering och superanrop i ett program som drar nytta av subtypspolymorfism
B5	Använda både överlagring och overriding på lämpligt sätt
B6	Förklara hur arvsbegreppet har använts i ett program för att separera genomskärande åtaganden
C7	Planering och uppföljning
C8	Visa hur man kan separera gränssnitt från implementation med hjälp av Java-interfaces
D9	Dokumentera icke-triviala modulers gränssnitt så att någon utomstående kan programmera mot dem
E10	Använda void-pekare i C-program för att uppnå genericitet på ett relevant sätt, t.ex. en datasamling som kan lagra godtyckligt data
E11	Använda parametrisk polymorfism för ökad "statisk typsäkerhet" vid interaktion med Javas standardbibliotek
E12	Designa med parametrisk polymorfism för att göra relevanta delar av ett program mer generellt
F13	Översätta mellan rekursiva och iterativa lösningar av samma problem samt diskutera för- och nackdelar med sido-effekter
F14	Förklara skillnaden mellan deklarativ och imperativ; diskutera deras respektive för- och nackdelar
G15	Använda kopiering eller liknande för att undvika aliaseringsproblem i ett C- eller Javaprogram
G16	Använda åtkomstmodifierare för att styra synligheten hos variabler och metoder
G17	Använda nästlade och inre klasser för att kapsla in privata beståndsdelar i ett sammansatt objekt
G18	Resonera kring hur inkapsling både kan försvåra och underlätta testning (ta gärna hjälp av kodexempel)
H19	Förklara skillnaden mellan identitet och ekvivalens
H20	Redovisat förståelse för skillnaden mellan ekvivalens och identitet

Kunskapsmål

Name	Short desc	Grade level
A1	Konsekvent tillämpa procedurell abstraktion för att öka läsbarheten och undvika upprepningar	3
A2	Tillämpa objektorienterad abstraktion för att dölja implementationsdetaljer bakom väldefinierade gränssnitt	3
A3	Demonstrera förståelse för designprincipen informationsgömning i ett C-program med hjälp av .c och .h-filer	4
B4	Använda arv, methodspecialisering och superanrop i ett program som drar nytta av subtypspolymorfism	3
B5	Använda både överlagring och overriding på lämpligt sätt	3
B6	Förklara hur arvsbegreppet har använts i ett program för att separera genomskärande åtaganden	4
C7	Planering och uppföljning	3
C8	Visa hur man kan separera gränssnitt från implementation med hjälp av Java-interfaces	4
D9	Dokumentera icke-triviala modulers gränssnitt så att någon utomstående kan programmera mot dem	3
E10	Använda void-pekare i C-program för att uppnå genericitet på ett relevant sätt, t.ex. en datasamling som kan lagra godtyckligt data	3
E11	Använda parametrisk polymorfism för ökad "statisk typsäkerhet" vid interaktion med Javas standardbibliotek	3
E12	Designa med parametrisk polymorfism för att göra relevanta delar av ett program mer generellt	4
F13	Översätta mellan rekursiva och iterativa lösningar av samma problem samt diskutera för- och nackdelar med sido-effekter	3
F14	Förklara skillnaden mellan deklarativ och imperativ; diskutera deras respektive för- och nackdelar	3
G15	Använda kopiering eller liknande för att undvika aliaseringsproblem i ett C- eller Javaprogram	3
G16	Använda åtkomstmodifierare för att styra synligheten hos variabler och metoder	3
G17	Använda nästlade och inre klasser för att kapsla in privata beståndsdelar i ett sammansatt objekt	4
G18	Resonera kring hur inkapsling både kan försvåra och underlätta testning (ta gärna hjälp av kodexempel)	5
H19	Förklara skillnaden mellan identitet och ekvivalens	3
H20	Redovisa förståelse för skillnaden mellan värdesemantik och referenssemantik med hjälp av ett kodexempel	3
H21	Redovisa förståelse för abstrakta klasser och metoder, samt deras relation till Java-interface	5
I22	Förklara innebörden av, och tillämpa på ett konsekvent sätt, defensiv programmering i ett program	3
I23	Fånga på lämpliga platser och hantera på ett lämpligt sätt kontrollerade och okontrollerade (relevanta) undantag i ett program	3
I24	Använd olika metoder för felhantering och relatera dem till varandra	4
I25	Utöver föregående mål, demonstrera fördjupad förståelse för undantagshantering i Java genom att kasta existerande och egendefinierade kontrollerade och okontrollerade undantag i ett program	4
J26	Demonstrera förståelse för skillnaden mellan allokering på stacken och allokering på heapen med hjälp av ett C-program	3
J27	Demonstrera förståelse för minneshantering i C genom att skriva ett program med dynamiska strukturer som är fritt från minnesläckage och argumentera för varför det är så (och verifiera med valgrind)	3
J28	Förklara skillnaden mellan C:s manuella minneshantering och hur Java hanterar minne och beskriv för ett lämpligt Java-program när minne allokeras och frigörs	4
J29	Jämför två metoder för automatisk skräpsamling	5
K30	Specificera tydliga gränssnitt mellan moduler i ett program som bör brytas ned i flera moduler (och implementera)	3
K31	Givet ett icke-trivialt program uppdelat i moduler som i ovanstående mål, resonera kring begreppen coupling och cohesion för några av modulerna	4
K32	Utveckla resonemanget i (4) med en diskussion om separation of concerns och hur detta har uppnåtts (alt. kan/inte kan) i programmet i fråga	5
L33	Parallellisera relevanta delar av ett tidigare program	3
L34	Utöver föregående mål, jämför den sekventiella och den parallella körtiden och förklara skillnaden	4
L35	Utöver föregående mål, justera eventuella parametrar (t.ex. sequential cutoff, antalet trådar) för att se hur prestanda påverkas, mät och förklara igen	5
M36	Översätta mellan C:s array-notation och pekararitmetik	3
M37	Använda pekare för att skapa länkade strukturer	3
M38	Använda pekare till stackvariabler för värdeöverföring mellan funktionsanrop	3
	program	4

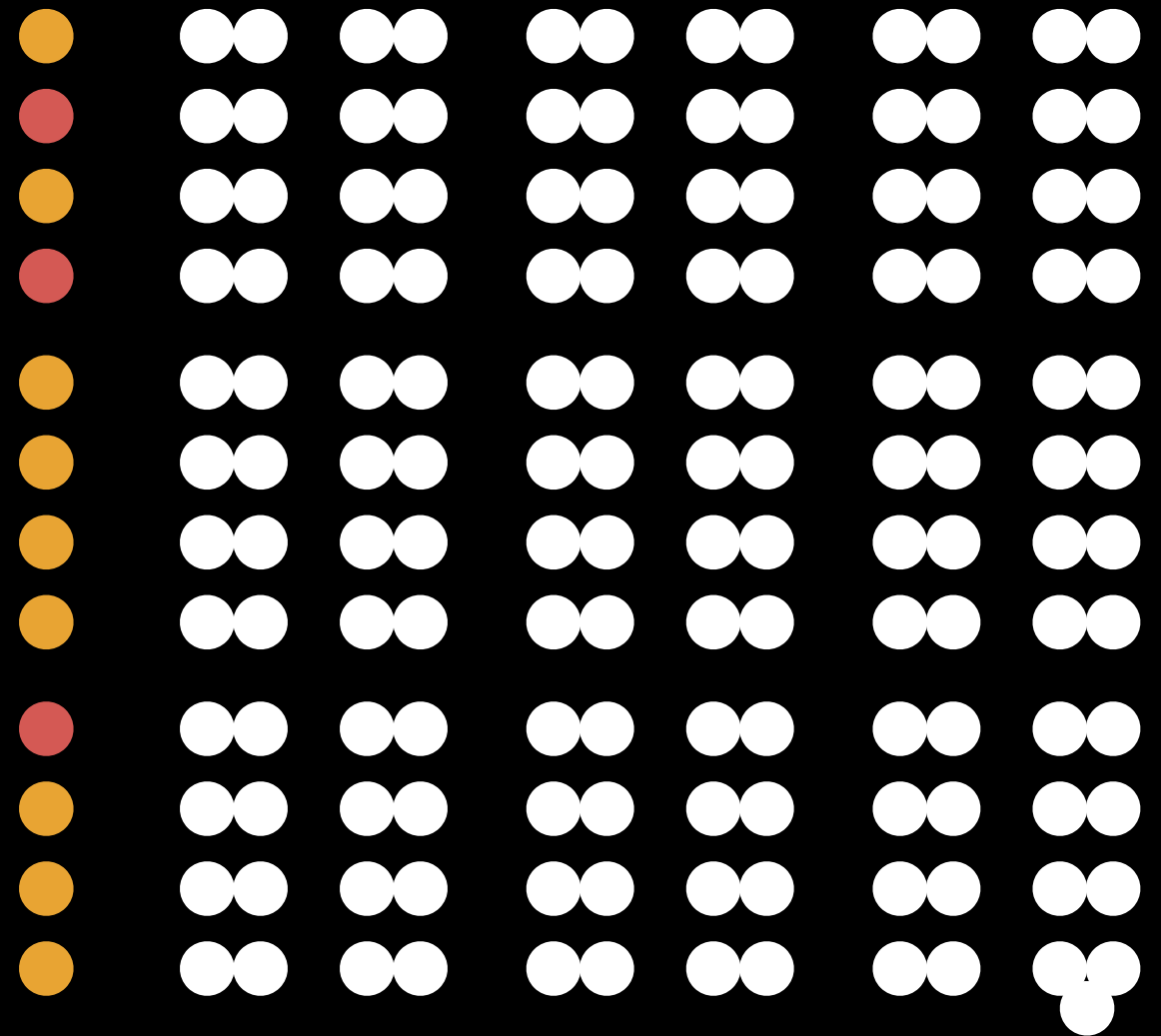
Kunskapsmål

Name	Short desc	Grade level
A1	Konsekvent tillämpa procedurrell abstraktion för att öka läsbarheten och undvika upprepningar	3
A2	Tillämpa objektorienterad abstraktion för att dölja implementationsdetaljer bakom väldefinierade gränssnitt	3
A3	Demonstrera förståelse för designprincipen informationsgömning i ett C-program med hjälp av .c och .h-filer	4
B4	Använda arv, methodspecialisering och superanrop i ett program som drar nytta av subtypspolymorfism	3
B5	Använda både överlagring och overriding på lämpligt sätt	3
B6	Förklara hur arvsbegreppet har använts i ett program för att separera genomskärande åtaganden	4
C7	Planering och uppföljning	3
C8	Visa hur man kan separera gränssnitt från implementation med hjälp av Java-interfaces	4
D9	Dokumentera icke-triviala modulers gränssnitt så att någon utomstående kan programmera mot dem	3
E10	Använda void-pekare i C-program för att uppnå genericitet på ett relevant sätt, t.ex. en datasamling som kan lagra godtyckligt data	3
E11	Använda parametrisk polymorfism för ökad "statisk typsäkerhet" vid interaktion med Javas standardbibliotek	3
E12	Designa med parametrisk polymorfism för att göra relevanta delar av ett program mer generellt	4
F13	Översätta mellan rekursiva och iterativa lösningar av samma problem samt diskutera för- och nackdelar med sido-effekter	3
F14	Förklara skillnaden mellan deklarativ och imperativ; diskutera deras respektive för- och nackdelar	3
G15	Använda kopiering eller liknande för att undvika aliaseringsproblem i ett C- eller Javaprogram	3
G16	Använda åtkomstmodifierare för att styra synligheten hos variabler och metoder	3
G17	Använda nästlade och inre klasser för att kapsla in privata beståndsdelar i ett sammansatt objekt	4
H19	Förklara skillnaden mellan identitet och ekvivalens	3
H20	Redovisa förståelse för skillnaden mellan värdesemantik och referenssemantik med hjälp av ett kodexempel	3
I22	Förklara innebörden av, och tillämpa på ett konsekvent sätt, defensiv programmering i ett program	3
I23	Fånga på lämpliga platser och hantera på ett lämpligt sätt kontrollerade och okontrollerade (relevanta) undantag i ett program	3
I24	Använd olika metoder för felhantering och relatera dem till varandra	4
I25	Utöver föregående mål, demonstrera fördjupad förståelse för undantagshantering i Java genom att kasta existerande och egendefinierade kontrollerade och okontrollerade undantag i ett program	4
J26	Demonstrera förståelse för skillnaden mellan allokering på stacken och allokering på heapen med hjälp av ett C-program	3
J27	Demonstrera förståelse för minneshantering i C genom att skriva ett program med dynamiska strukturer som är fritt från minnesläckage och argumentera för varför det är så (och verifiera med valgrind)	3
J28	Förklara skillnaden mellan C:s manuella minneshantering och hur Java hanterar minne och beskriv för ett lämpligt Java-program när minne allokeras och frigörs	4
K30	Specificera tydliga gränssnitt mellan moduler i ett program som bör brytas ned i flera moduler (och implementera)	3
K31	Givet ett icke-trivialt program uppdelat i moduler som i ovanstående mål, resonera kring begreppen coupling och cohesion för några av modulerna	4
L33	Parallellisera relevanta delar av ett tidigare program	3
L34	Utöver föregående mål, jämför den sekventiella och den parallella körtiden och förklara skillnaden	4
M36	Översätta mellan C:s array-notation och pekararitmetik	3
M37	Använda pekare för att skapa länkade strukturer	3
M38	Använda pekare till stackvariabler för värdeöverföring mellan funktionsanrop	3
herokuapp.com/achievements/21		4

Kunskapsmål

Name	Short desc	Grade level
A1	Konsekvent tillämpa procedurell abstraktion för att öka läsbarheten och undvika upprepningar	3
A2	Tillämpa objektorienterad abstraktion för att dölja implementationsdetaljer bakom väldefinierade gränssnitt	3
B4	Använda arv, metodspecialisering och superanrop i ett program som drar nytta av subtypspolymorfism	3
B5	Använda både överlagring och overriding på lämpligt sätt	3
C7	Planering och uppföljning	3
D9	Dokumentera icke-triviala modulers gränssnitt så att någon utomstående kan programmera mot dem	3
E10	Använda void-pekare i C-program för att uppnå genericitet på ett relevant sätt, t.ex. en datasamling som kan lagra godtyckligt data	3
E11	Använda parametrisk polymorfism för ökad "statisk typsäkerhet" vid interaktion med Javas standardbibliotek	3
F13	Översätta mellan rekursiva och iterativa lösningar av samma problem samt diskutera för- och nackdelar med sido-effekter	3
F14	Förklara skillnaden mellan deklarativ och imperativ; diskutera deras respektive för- och nackdelar	3
G15	Använda kopiering eller liknande för att undvika aliaseringsproblem i ett C- eller Javaprogram	3
G16	Använda åtkomstmodifierare för att styra synligheten hos variabler och metoder	3
H19	Förklara skillnaden mellan identitet och ekvivalens	3
H20	Redovisa förståelse för skillnaden mellan värdesemantik och referenssemantik med hjälp av ett kodexempel	3
I22	Förklara innebörden av, och tillämpa på ett konsekvent sätt, defensiv programmering i ett program	3
I23	Fånga på lämpliga platser och hantera på ett lämpligt sätt kontrollerade och okontrollerade (relevanta) undantag i ett program	3
J26	Demonstrera förståelse för skillnaden mellan allokering på stacken och allokering på heapen med hjälp av ett C-program	3
J27	Demonstrera förståelse för minneshantering i C genom att skriva ett program med dynamiska strukturer som är fritt från minnesläckage och argumentera för varför det är så (och verifiera med valgrind)	3
K30	Specificera tydliga gränssnitt mellan moduler i ett program som bör brytas ned i flera moduler (och implementera)	3
L33	Parallellisera relevanta delar av ett tidigare program	3
M36	Översätta mellan C:s array-notation och pekararitmetik	3
M37	Använda pekare för att skapa länkade strukturer	3
M38	Använda pekare till stackvariabler för värdeöverföring mellan funktionsanrop	3

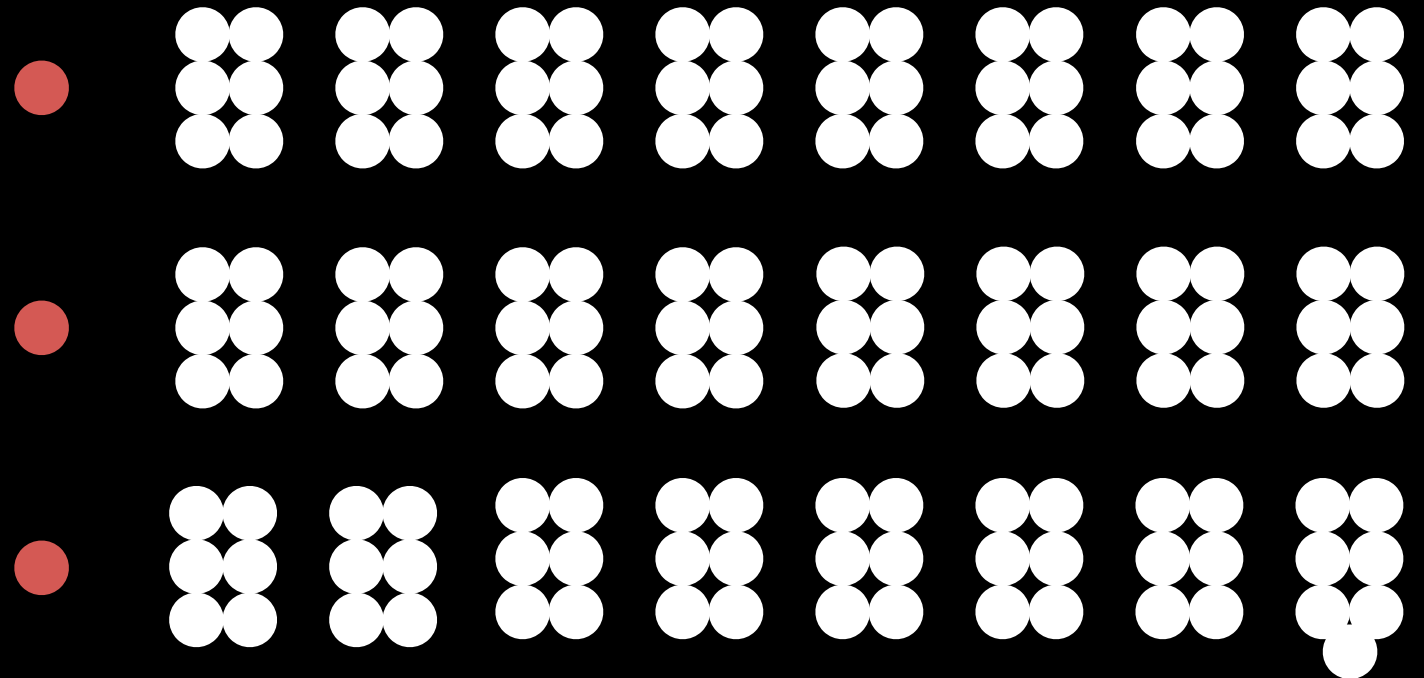
Phase 2: Project



Phase 2: Project

We split them
into groups of ~6.

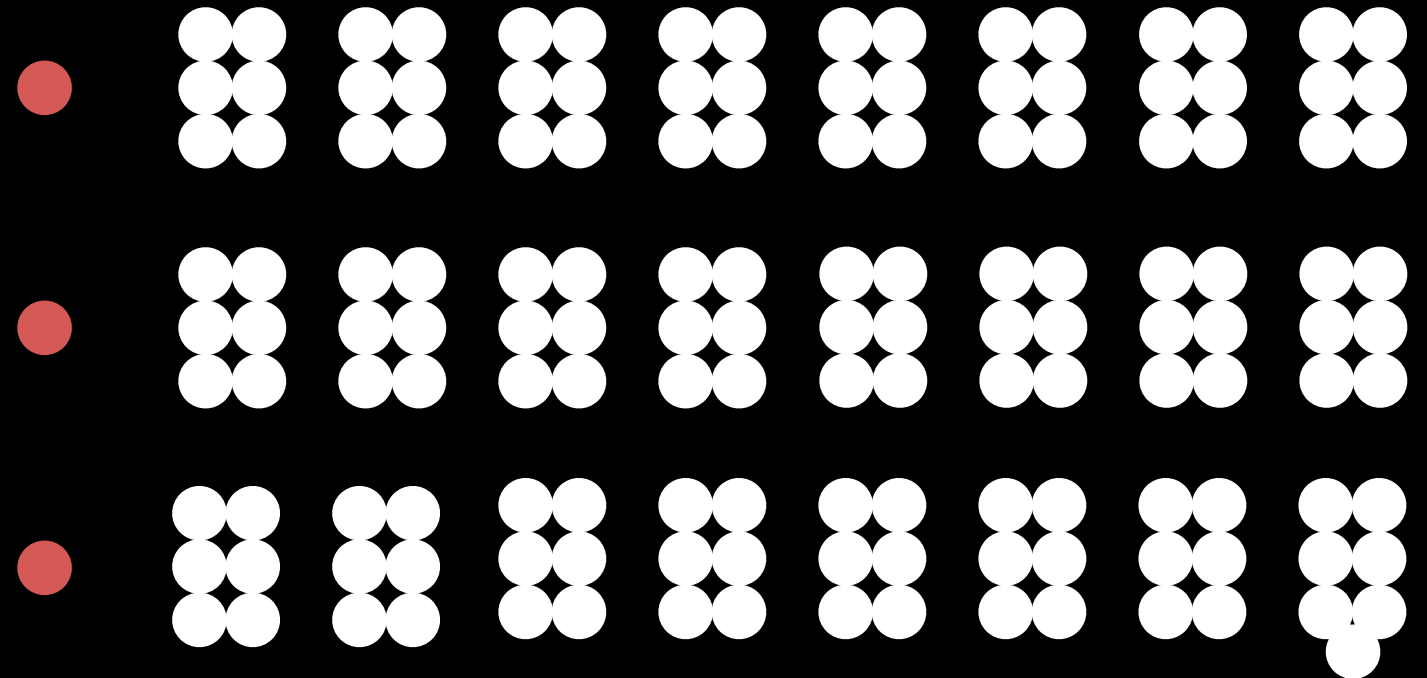
Each group gets a
supervisor assigned.



Phase 2: Project

We split them
into groups of ~6.

Each group gets a
supervisor assigned.



Groups meet their supervisor “regularly”.

Phase 2: Project

1

INL

Really hard!

Projektarbetet består av en specifikation (detta dokument) som skall implementeras i programspråket C. Arbetet skall utföras i team om 6 personer som jobbar i roterande par. Vårt mål är att två team skall bilda en grupp som kommer att redovisa samtidigt. Denna koordination sköts internt av kursledningen.

Projektet har många syften: att fördjupa kunskaperna i C, att bygga programmeringserfarenhet på djupet och bygga på den kunskap som byggts upp under föregående faser, samt att ge en plattform för ytterligare redovisning av mål inom ramarna för ett "riktigt program". Ett av huvudsyftena är också att introducera element från *programvarutekniken*, d.v.s. den ingenjörsciensdisciplin som sysslar med utveckling av mjukvara inom givna tids-, kostnads- och kvalitetsramar och här specifikt testning. Genom att utföra en litet större uppgift mellan par som utför olika delar av uppgiften, kommer ni att få uppleva vikten av klart definierade processer och roller och klart definierade gränssnitt mellan programmoduler. Målet är inte "att visa hur man gör", utan snarare att försöka ge en bakgrund till varför metoder och tekniker som tas upp på senare kurser är nödvändiga för systematisk utveckling av mjukvara.

1.1 Processen

Arbetet skall utföras i team om 6 personer updelade i 3 programmeringsgrupper, uppdelade i två team. Arbetet del...

WORK BREAKDOWN STRUCTURE

1. Se till att sätta teamet i samband
2. Läs dessa instruktioner noggrant, sedan en gång till
3. Löpande under projektet, dokumentera & versionshantera
4. Planering och design (≈ 2 dgr)
 - (a) Gör en övergripande design för systemet
 - (b) Dela upp systemet i lika många "delsystem" som det finns par
 - (c) Definiera gränssnitten mellan delsystemen
5. Implementation, parallellt i paren (> 2 v)
 - (a) Dela upp ditt delsystem i delar Δ, Δ', \dots
 - (b) Fundera ut hur man testar del Δ
 - (c) Implementera testerna T_Δ för Δ ,
 - (d) Implementera Δ och testa löpande mot T_Δ
(Upprepa steg b-d...)
6. Integration (≈ 1 v)
 - (a) Sätt ihop delsystemen, testa, åtgärda fel
7. Reflektera kort över projektet

Phase 2: Project

1 | INL *Really hard!*

Projektarbetet består av en specifikation (detta dokument) som skall implementeras i programspråket C. Arbetet skall utföras i team om 6 personer som jobbar i roterande par. Vårt mål är att två team skall bilda en grupp som kommer att redovisa samtidigt. Denna koordination sköts internt av kursledningen.

Projektet har många syften: att fördjupa kunskaperna i C, att bygga programmeringserfarenhet på djupet och bygga på den kunskap som byggts upp under föregående faser, samt att ge en plattform för ytterligare redovisning av mål inom ramarna för ett "riktigt program". Ett av huvudsyftena är också att introducera element från *programvarutekniken*, d.v.s. den ingenjörsciensdisciplin som sysslar med utveckling av mjukvara inom givna tids-, kostnads- och kvalitetsramar och här specifikt testning. Genom att utföra en litet större uppgift än enkla laborationer och inlämningsuppgifter, och införa samarbetsmoment mellan par som utför olika delar av uppgiften, kommer ni att få uppleva vikten av klart definierade processer och roller och klart definierade gränssnitt mellan programmoduler. Målet är inte "att visa hur man gör", utan snarare att försöka ge en bakgrund till varför metoder och tekniker som tas upp på senare kurser är nödvändiga för systematisk utveckling av mjukvara.

1.1 Processen

Arbetet skall utföras i team om 6 personer uppdelade i 3 programmeringsgrupper uppdelade i två team. Arbetet del...

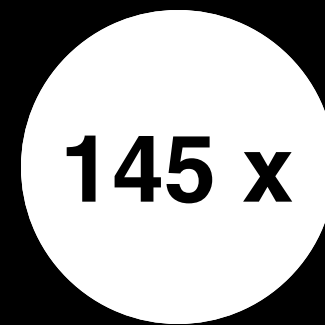
WORK BREAKDOWN STRUCTURE

1. Se till att sätta teamet i samband
2. Läs dessa instruktioner noggrant, sedan en gång till
3. Löpande under projektet, dokumentera & versionshantera
4. Planering och design (≈ 2 dgr)
 - (a) Gör en övergripande design för systemet
 - (b) Dela upp systemet i lika många "delsystem" som det finns par
 - (c) Definiera gränssnitten mellan delsystemen
5. Implementation, parallellt i paren (> 2 v)
 - (a) Dela upp ditt delsystem i delar Δ, Δ', \dots
 - (b) Fundera ut hur man testar del Δ
 - (c) Implementera testerna T_Δ för Δ ,
 - (d) Implementera Δ och testa löpande mot T_Δ
(Upprepa steg b-d...)
6. Integration (≈ 1 v)
 - (a) Sätt ihop delsystemen, testa, åtgärda fel
7. Reflektera kort över projektet


Chance to apply.
Chance to understand how much they don't know yet.

Part II
Getting Feedback!

Three Sources of Feedback



Three Sources Feedback



Discussions before+after+during course



Regular TA lunches before+during course



Lots!

Three Sources Feedback

- Lab sessions — oral examination works wonders!
 - Course council — small group meets teacher
 - Group meetings — PhD student TAs meet students
 - Web forum — students ask questions, comment, and complain!
 - Questionnaires — comprehensive survey in last third of course
-

Feedback we've gotten

- Lab: Many students have similar misunderstanding.
- Reaction: Tell teacher; will be an extra slide next day.

Feedback we've gotten

- Course council: “Too much to do! Conflict with other course’s assignment!”
- Reaction: Dropped an assignment

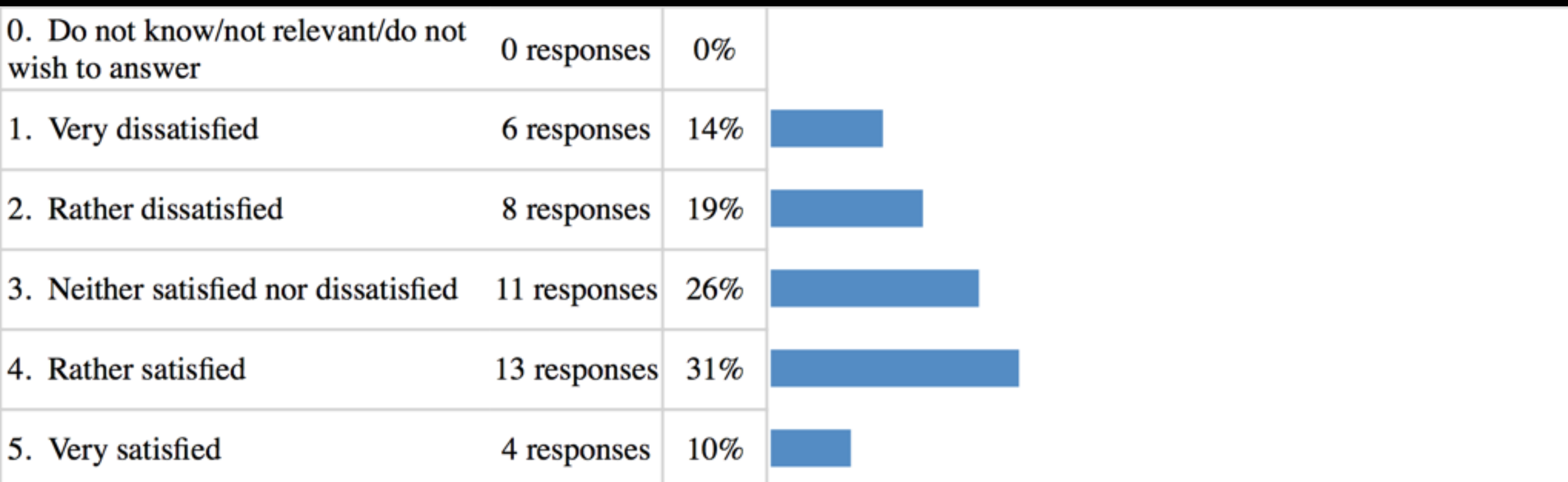
Feedback we've gotten

- Group meetings: Most students struggle with pointers (a tough concept)
- Reaction: Offer extra small group sessions on pointers.

Feedback we've gotten

- Questionnaires: students **really** appreciate how much we try to make the course better.
- Reaction: Keep doing that :)

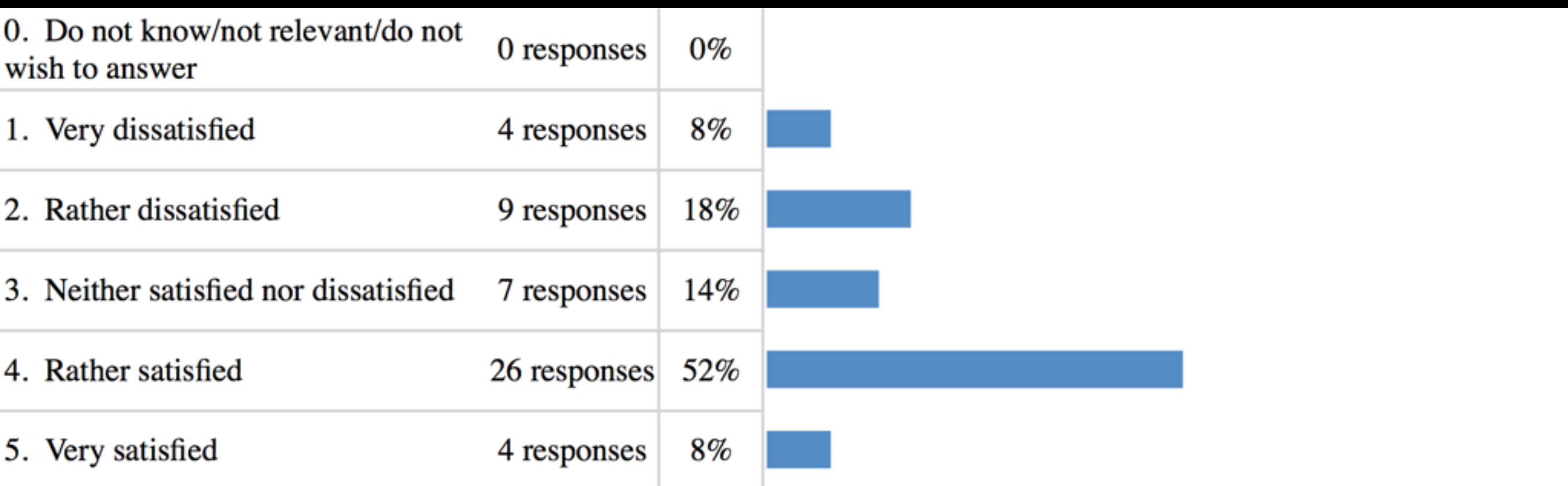
"Overall, how satisfied are you with the course?"



Mean: 3.0/5

2013 - the chaotic 1st year

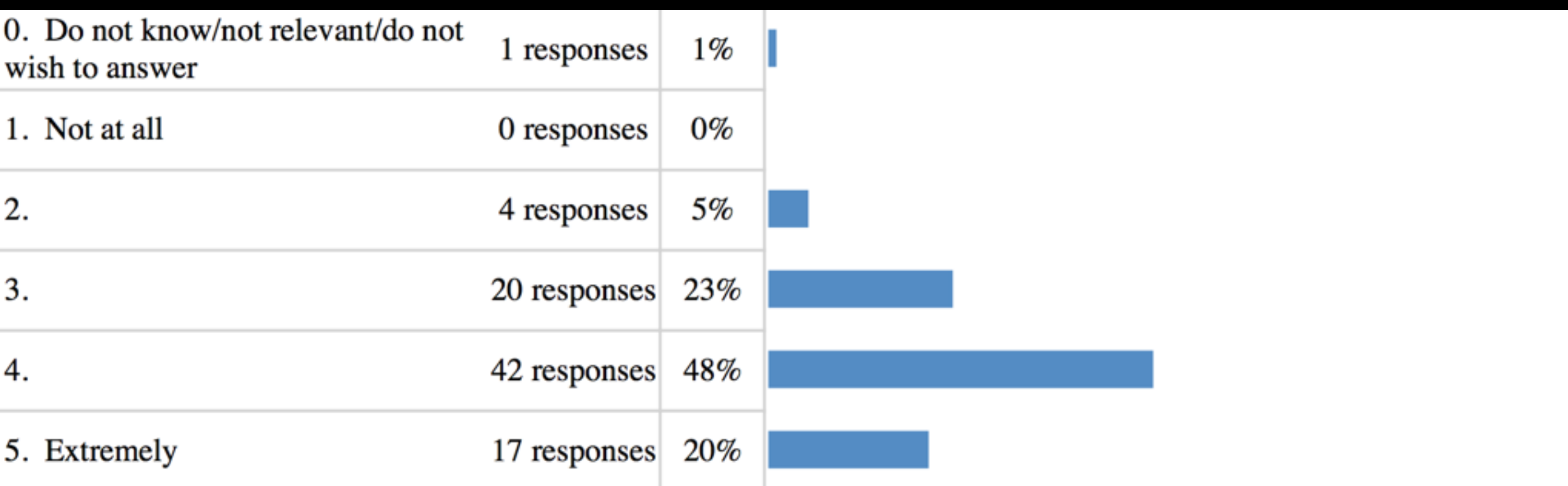
"Overall, how satisfied are you with the course?"



Mean: 3.3/5

2014 - learn from past mistakes

"Overall, how satisfied are you with the course?"



Mean: 3.9/5

2015 - some more polishing!

145 students — 12 TAs — 1 Teacher

Feedback drives the design of the course.

Lots of work — but gratifying.

No grading papers!

From 3.0/5 to 3.9/5 satisfaction in 3 years.