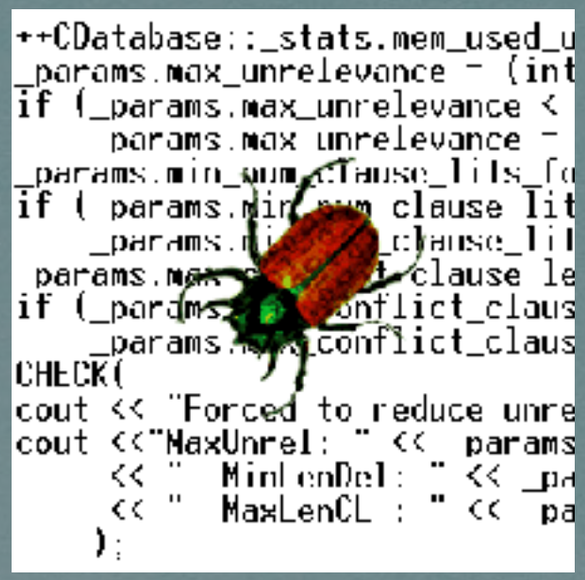
مقدمه‌ای بر آزمون نرم‌افزار

محمد تنهایی

سه بخش اصلی بحث

1. چرا باید آزمون انجام دهیم؟
2. چه کاری در زمان آزمون باید انجام دهیم؟
3. چگونه آزمون را به تئوری‌های کنونی نزدیک کنیم؟

Hopper's bug
(حشره‌ای در
اولین ماشین
ساخته شده)



"an analyzing process must equally have been performed in order to furnish the Analytical Engine with the necessary operative data; and that herein may also lie a possible source of error. Granted that the actual mechanism is unerring in its processes, the cards may give it wrong orders. " – Ada, Countess Lovelace (notes on Babbage's Analytical Engine)

"It has been just so in all of my inventions. The first step is an intuition, and comes with a burst, then difficulties arise—this thing gives out and *[it is]* then that 'Bugs'—as such little faults and difficulties are called—show themselves and months of intense watching, study and labor are requisite. . ." – Thomas Edison

اشکال در محصول نرم‌افزار

- **مریخ‌پیمای ناسا**: در سپتامبر ۱۹۹۹، خطای مربوط به یکپارچگی، ۵۰ میلیون دلار خسارت
- خسارت بزرگ با توجه به خطاهای ابزارهای تحت وب
  - **سرویس‌های مالی**: هر ساعت ۶/۵ میلیون دلار
  - **پرداخت برخط**: ۲/۴ میلیون دلار هر یک ساعت
- در دسامبر ۲۰۰۶ ، **فروش BOGO** در آمازون با دو بار تخفیف انجام شد.
- در سال ۲۰۰۷، سمانتیک اعلام کرد که بخش بزرگی از **حفره‌های امنیتی** ناشی از اشکال در نرم‌افزار است
- **آزمون‌های قوی‌تر** می‌تواند اشکالات نرم‌افزار را پیدا کند.

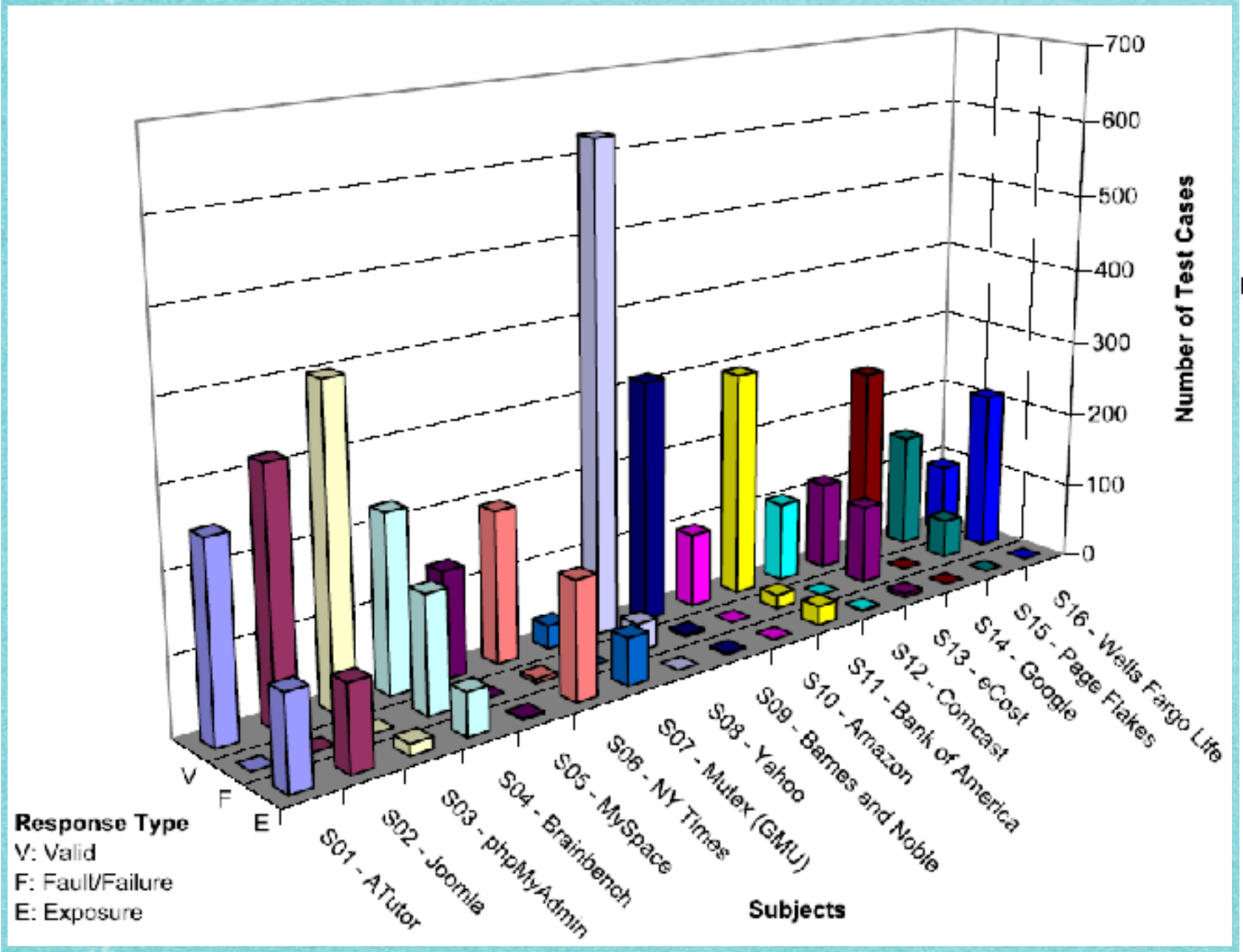
اشکال در محصول نرم‌افزار

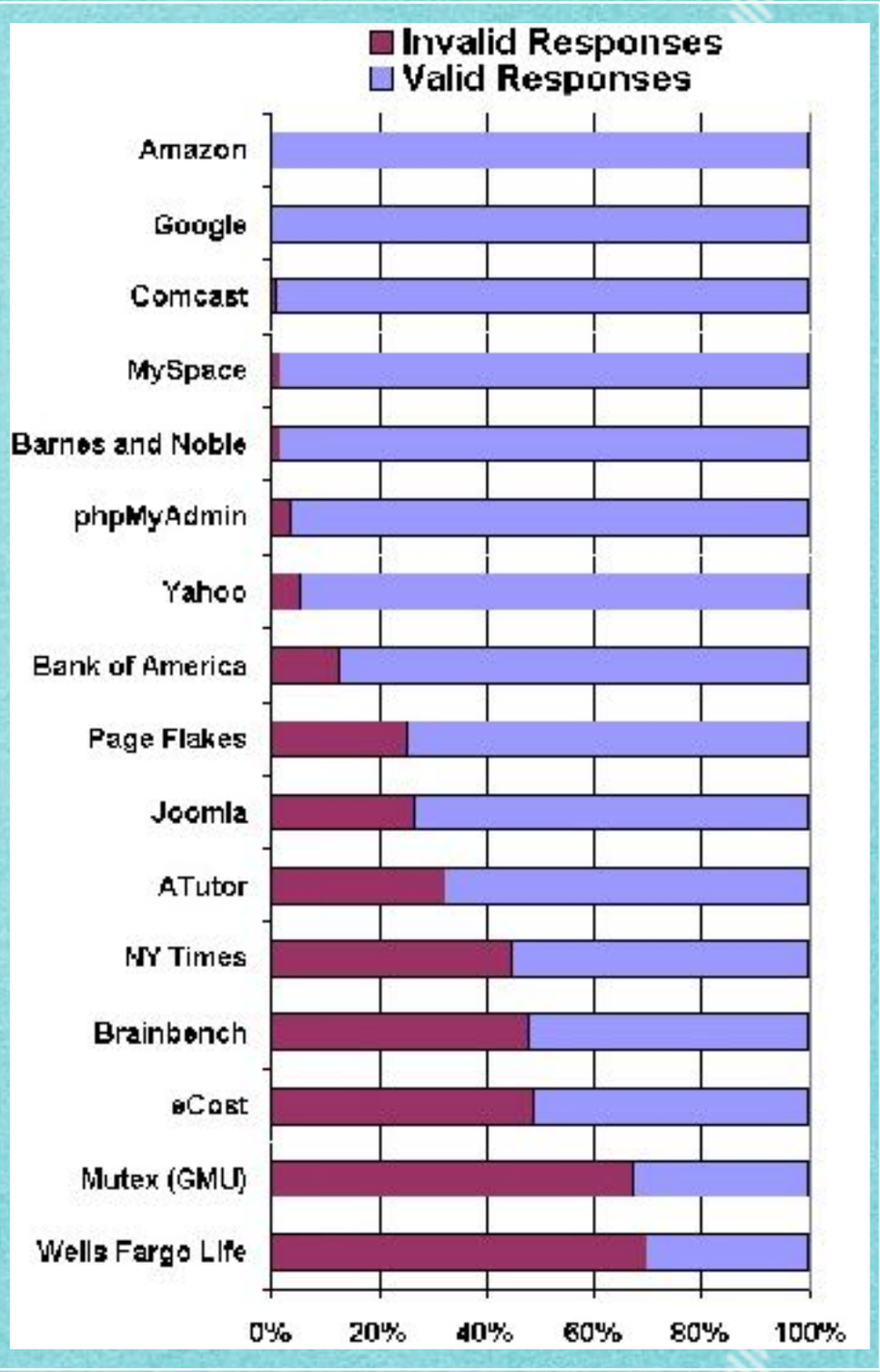
- **مریخ‌پیمای ناسا**: در سپتامبر ۱۹۹۹، خطای مربوط به یکپارچگی، ۵۰ میلیون دلار خسارت
- خسارت بزرگ با توجه به خطاهای ابزارهای تحت وب

خسارت مالی بر اساس اشکال نرم‌افزارها هر سال تکرار می‌شود

- **پرداخت برخط**: ۲/۴ میلیون دلار هر یک ساعت
- در دسامبر ۲۰۰۶ ، **فروش BOGO** در آمازون با دو بار تخفیف انجام شد.
- در سال ۲۰۰۷، سمانتیک اعلام کرد که بخش بزرگی از **حفره‌های امنیتی** ناشی از اشکال در نرم‌افزار است
- **آزمون‌های قوی‌تر** می‌تواند اشکالات نرم‌افزار را پیدا کند.

Bypass Testing Results





— Vasileios Papadimitriou. Masters thesis, *Automating Bypass Testing for Web Applications*, GMU 2006

چرا آزمون نرم‌افزار مهم است؟

چرا آزمون نرم‌افزار مهم است؟

- گزارش سال NIST ۲۰۰۲ : میزان تاثیر عدم وجود زیرساخت آزمون نرم‌افزار
- میزان خسارت در امریکا: بین ۲۲ تا ۵۹ میلیارد دلار
- روش‌های مناسب می‌تواند هزینه را نصف کند

چرا آزمون نرم‌افزار مهم است؟

- گزارش سال NIST ۲۰۰۲ : میزان تاثیر عدم وجود زیرساخت آزمون نرم‌افزار
- میزان خسارت در امریکا: بین ۲۲ تا ۵۹ میلیارد دلار
- روش‌های مناسب می‌تواند هزینه را نصف کند
- فجایای بزرگ: آریان ۵، اشکال مریخ‌نورد، باگ پنتیوم FDIV

چرا آزمون نرم‌افزار مهم است؟

- گزارش سال NIST ۲۰۰۲ : میزان تاثیر عدم وجود زیرساخت آزمون نرم‌افزار
  - میزان خسارت در امریکا: بین ۲۲ تا ۵۹ میلیارد دلار
  - روش‌های مناسب می‌تواند هزینه را نصف کند
- فجایای بزرگ: آریان ۵، اشکال مریخ‌نورد، باگ پنتیوم FDIV
- عدم آزمون نرم‌افزارهای مربوط به سلامت:
  - THERAC-25 ، سبب مرگ سه نفر (تشعشع رادیویی)
- نیاز به طراحی نرم‌افزارهایی با قابلیت اطمینان داریم

چرا آزمون نرم‌افزار مهم است؟

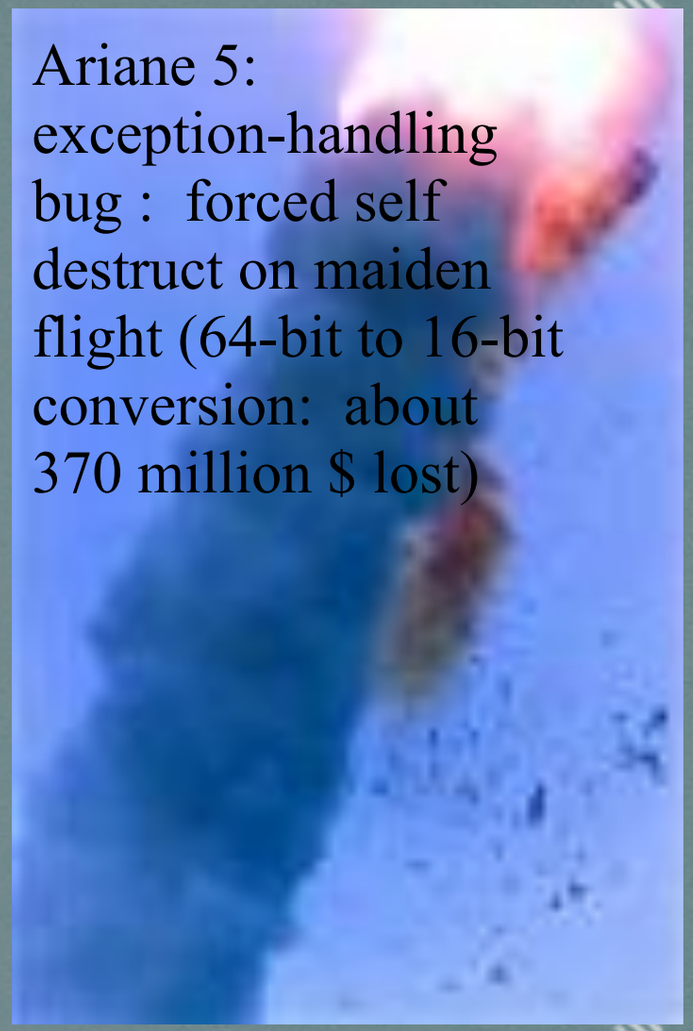
- گزارش سال NIST ۲۰۰۲ : میزان تاثیر عدم وجود زیرساخت آزمون نرم‌افزار
  - میزان خسارت در امریکا: بین ۲۲ تا ۵۹ میلیارد دلار
  - روش‌های مناسب می‌تواند هزینه را نصف کند
- فجایای بزرگ: آریان ۵، اشکال مریخ‌نورد، باگ پنتیوم FDIV
- عدم آزمون نرم‌افزارهای مربوط به سلامت:
  - THERAC-25 ، سبب مرگ سه نفر (تشعشع رادیویی)
- نیاز به طراحی نرم‌افزارهایی با قابلیت اطمینان داریم
  - آزمون یکی از روش‌های اطمینان از درستی عملکرد نرم‌افزارها است.

چرا آزمون نرم‌افزار مهم است؟

- گزارش سال NIST ۲۰۰۲ : میزان تاثیر عدم وجود زیرساخت آزمون نرم‌افزار
  - میزان خسارت در امریکا: بین ۲۲ تا ۵۹ میلیارد دلار
  - روش‌های مناسب می‌تواند هزینه را نصف کند
- فجایای بزرگ: آریان ۵، اشکال مریخ‌نورد، باگ پنتیوم FDIV
- عدم آزمون نرم‌افزارهای مربوط به سلامت:
  - THERAC-25 ، سبب مرگ سه نفر (تشعشع رادیویی)
- نیاز به طراحی نرم‌افزارهایی با قابلیت اطمینان داریم
  - آزمون یکی از روش‌های اطمینان از درستی عملکرد نرم‌افزارها است.

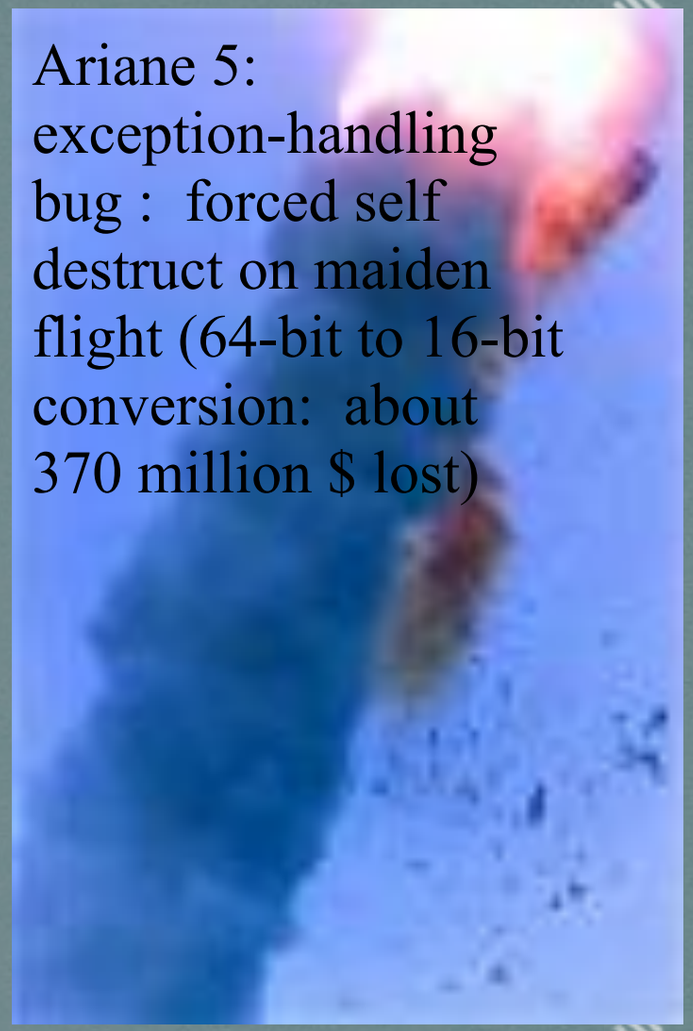
چرا آزمون نرم‌افزار مهم است؟

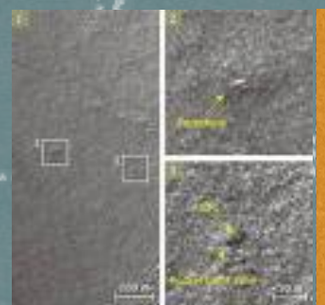
- گزارش سال NIST ۲۰۰۲ : میزان تاثیر عدم وجود زیرساخت آزمون نرم‌افزار
- میزان خسارت در امریکا: بین ۲۲ تا ۵۹ میلیارد دلار
- روش‌های مناسب می‌تواند هزینه را نصف کند
- فجایای بزرگ: آریان ۵، اشکال مریخ‌نورد، باگ پنتیوم FDIV
- عدم آزمون نرم‌افزارهای مربوط به سلامت:
  - THERAC-25 ، سبب مرگ سه نفر (تشعشع رادیویی)
- نیاز به طراحی نرم‌افزارهایی با قابلیت اطمینان داریم
  - آزمون یکی از روش‌های اطمینان از درستی عملکرد نرم‌افزارها است.



چرا آزمون نرم‌افزار مهم است؟

- گزارش سال NIST ۲۰۰۲ : میزان تاثیر عدم وجود زیرساخت آزمون نرم‌افزار
- میزان خسارت در امریکا: بین ۲۲ تا ۵۹ میلیارد دلار
- روش‌های مناسب می‌تواند هزینه را نصف کند
- فجایای بزرگ: آریان ۵، اشکال مریخ‌نورد، باگ پنتیوم FDIV
- عدم آزمون نرم‌افزارهای مربوط به سلامت:
  - THERAC-25 ، سبب مرگ سه نفر (تشعشع رادیویی)
- نیاز به طراحی نرم‌افزارهایی با قابلیت اطمینان داریم
  - آزمون یکی از روش‌های اطمینان از درستی عملکرد نرم‌افزارها است.

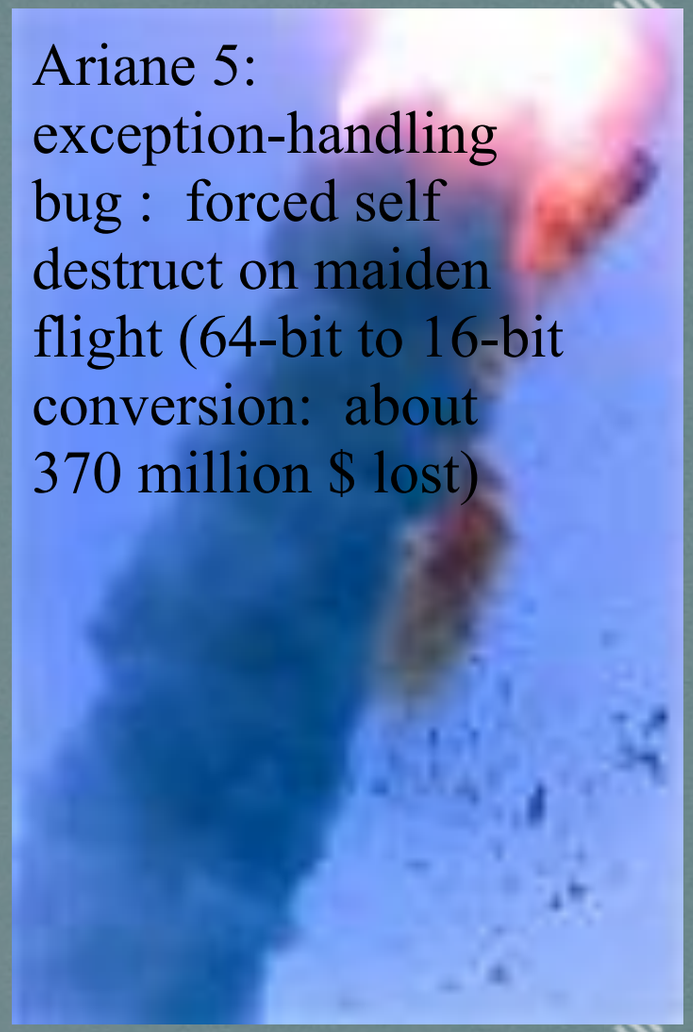


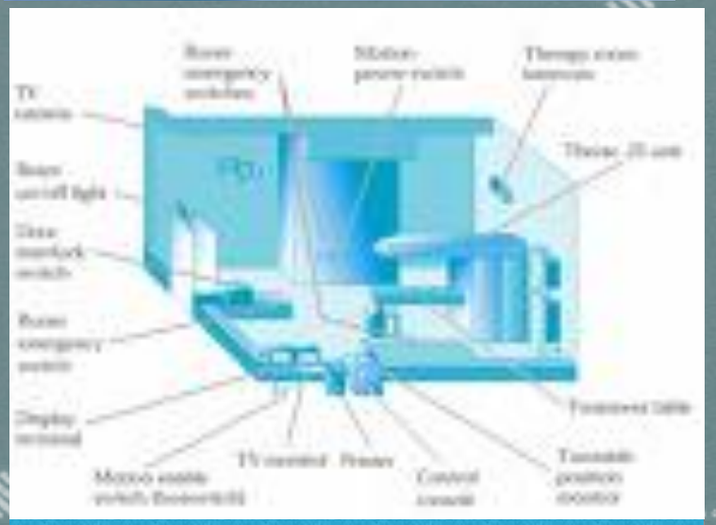


Mars Polar Lander crash site?

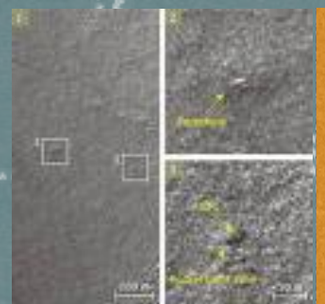
چرا آزمون نرم‌افزار مهم است؟

- گزارش سال ۲۰۰۲ NIST : میزان تاثیر عدم وجود زیرساخت آزمون نرم‌افزار
- میزان خسارت در امریکا: بین ۲۲ تا ۵۹ میلیارد دلار
- روش‌های مناسب می‌تواند هزینه را نصف کند
- فجایای بزرگ: آریان ۵، اشکال مریخ‌نورد، باگ پنتیوم FDIV
- عدم آزمون نرم‌افزارهای مربوط به سلامت:
  - THERAC-25 ، سبب مرگ سه نفر (تشعشع رادیویی)
- نیاز به طراحی نرم‌افزارهایی با قابلیت اطمینان داریم
  - آزمون یکی از روش‌های اطمینان از درستی عملکرد نرم‌افزارها است.





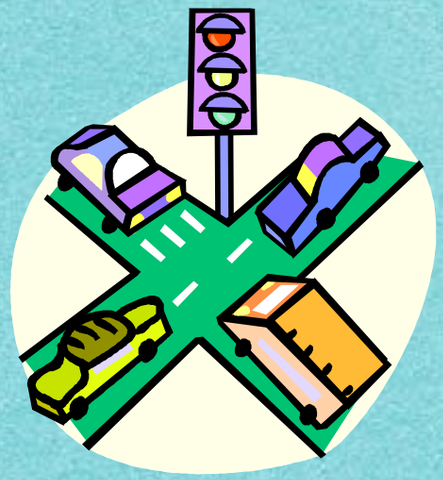
THERAC-25 design

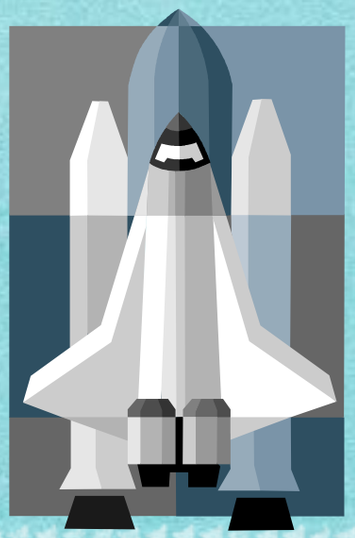


Mars Polar Lander crash site?

نرم‌افزار مانند یک پوسته اطراف
تمدن بشری را پوشانده است

نرم‌افزار مانند یک پوسته اطراف تمدن بشری را پوشانده است





نرم‌افزار کنترل شرایط بحرانی
ایرباس ۳۱۹

نرم‌افزار کنترل شرایط بحرانی ایرباس ۳۱۹

از دست رفتن پرواز خودکار

نرم‌افزار کنترل شرایط بحرانی ایرباس ۳۱۹

از دست رفتن پرواز خودکار

از دست رفتن ارتباط رادیویی و هشدار دهنده‌ها

نرم‌افزار کنترل شرایط بحرانی ایرباس ۳۱۹

از دست رفتن پرواز خودکار

از دست رفتن ارتباط رادیویی و هشدار دهنده‌ها

از دست رفتن خلبان و کمک خلبان و مانیتورهای مربوط به کنترل و هدایت پرواز

خاموشی سال ۲۰۰۳

خاموشی سال ۲۰۰۳

خاموشی سال ۲۰۰۳

508 generating units and 256 power plants shut down

خاموشی سال ۲۰۰۳

508 generating units and 256 power plants shut down

Affected 10 million people in Ontario, Canada

خاموشی سال ۲۰۰۳

508 generating units and 256 power plants shut down

Affected 10 million people in Ontario, Canada

Affected 40 million people in 8 US states

خاموشی سال ۲۰۰۳

508 generating units and 256 power plants shut down

Affected 10 million people in Ontario, Canada

Affected 40 million people in 8 US states

Financial losses of $6 Billion USD

خاموشی سال ۲۰۰۳

508 generating units and 256 power plants shut down

Affected 10 million people in Ontario, Canada

Affected 40 million people in 8 US states

Financial losses of $6 Billion USD

سیستم آلارم در سیستم مدیریت انرژی دچار اشکال می‌شود. اپراتور انسانی متوجه overload برق در سیستم انتقال نمی‌شود.

آزمون نرم‌افزاری روزبه‌روز مهم‌تر می‌شود

آزمون نرم‌افزاری در قرن ۲۱

- در روزگار کنونی همه چیز در حال تغییر است!
- نرم‌افزار رفتارها را تعریف می‌کند
  - روترهای شبکه، تجارت، سوییچ‌های شبکه و زیرساخت‌های دیگر
- بازار کنونی نرم‌افزار:
  - بسیار بزرگ‌تر
  - بسیار رقابتی‌تر
  - دارای کاربران بیشتر
- روش‌های چابک فشار زیادی روی آزمون می‌آورند.

- airplanes, air traffic control
- spaceships
- watches
- ovens
- remote controllers
- PDAs
- memory seats
- DVD players
- garage door openers
- cell phones

آزمون نرم‌افزاری در قرن ۲۱

- در روزگار کنونی همه چیز در حال تغییر است!
- نرم‌افزار رفتارها را تعریف می‌کند
- روترهای شبکه،
  زیرساخت‌های د
- بازار کنونی نرم‌افزار
- بسیار بزرگ‌تر
- بسیار رقابتی‌تر
- دارای کاربران بیشتر
- روش‌های چابک فشار زیادی روی آزمون می‌آورند.

- airplanes, air traffic control
- spaceships
- watches
- ovens
- remote controller
- PDAs
- memory seats
- DVD players
- garage door openers
- cell phones

نقش آزمون نرم‌افزاری در موفقیت نرم‌افزارها و صنایع وابسته در حال تغییر است.

آزمون نرم‌افزاری در قرن ۲۱

- نرم‌افزارهای حیاتی و زمان واقعی بیشتر
- کاربردهای تجاری به معنای نرم‌افزارهای بسیار بزرگ است.
- نرم‌افزارهای تعبیه شده همه جا هستند!
- نرم‌افزارهای مجانی انتظارات را از نرم‌افزارهای تجاری بالاتر برده است.
- امنیت در حال حاضر بیشتر به معنی نبود اشکال نرم‌افزاری است
  - نرم‌افزار امن نرم‌افزار با قابل اطمینان بالا است.

- اینترنت همه چیز را تغییر داده است
  - فضای رقابتی
  - پراکندگی بالا
  - نیاز به قابلیت اطمینان بسیار بالا

آزمون نرم‌افزاری در قرن ۲۱

- نرم‌افزارهای حیاتی و زمان واقعی بیشتر
- کاربردهای تجاری به معنای نرم‌افزارهای بسیار بزرگ است.
- نرم‌افزارهای تعبیه شده همه جا هستند!
- نرم‌افزارهای مجانی انتظارات را از نرم‌افزارهای تجاری بالاتر برده است.
- امنیت در حال حاضر بیشتر به معنی نبود اشکال نرم‌افزاری است

صنعت تشنه روش‌های ما برای آزمون است!

- اینترنت همه چیز را تغییر داده است
  - فضای رقابتی
  - پراکندگی بالا
  - نیاز به قابلیت اطمینان بسیار بالا

اشتباه در نیاز و هدف

- صنعت نیاز به آزمون‌های ساده و راحت دارد!
  - آزمونگرهایی که سر رشته‌ای از ریاضیات و محاسبات ندارند
  - دانشگاه‌ها دانشمندان مختلفی را فارغ التحصیل می کنند
  - صنعت نیاز به مهندس دارد
- آزمون باید با دقت بیشتری انجام شود
- فرآیندهای چابک خواسته‌های زیادی از آزمون دارد:
  - برنامه نویسان باید unit test انجام دهند. (بدون آموزش و داشتن ابزار و علم آن!)
  - آزمون یکی از مولفه‌های اصلی فرآیند است! اما چه کسی آنرا می‌سازد؟

اشتباه در نیاز و هدف

- صنعت نیاز به آزمون‌های ساده و راحت دارد!
  - آزمونگرهایی که سر رشته‌ای از ریاضیات و محاسبات ندارند
  - دانشگاه‌ها دانشمندان مختلفی را فارغ التحصیل می کنند

نتیجه: دسته زیادی از نرم‌افزارهای به درد نخور!

- فرآیندهای چابک خواسته‌های زیادی از آزمون دارد:
  - برنامه نویسان باید unit test انجام دهند. (بدون آموزش و داشتن ابزار و علم آن!)
  - آزمون یکی از مولفه‌های اصلی فرآیند است! اما چه کسی آنرا می‌سازد؟

بیاید! تحویل شما!

اولین کار **حرفه‌ای** من!

یک پشته از برگه‌های پرینت شده! بدون هیچ راهنما یا فهرستی!

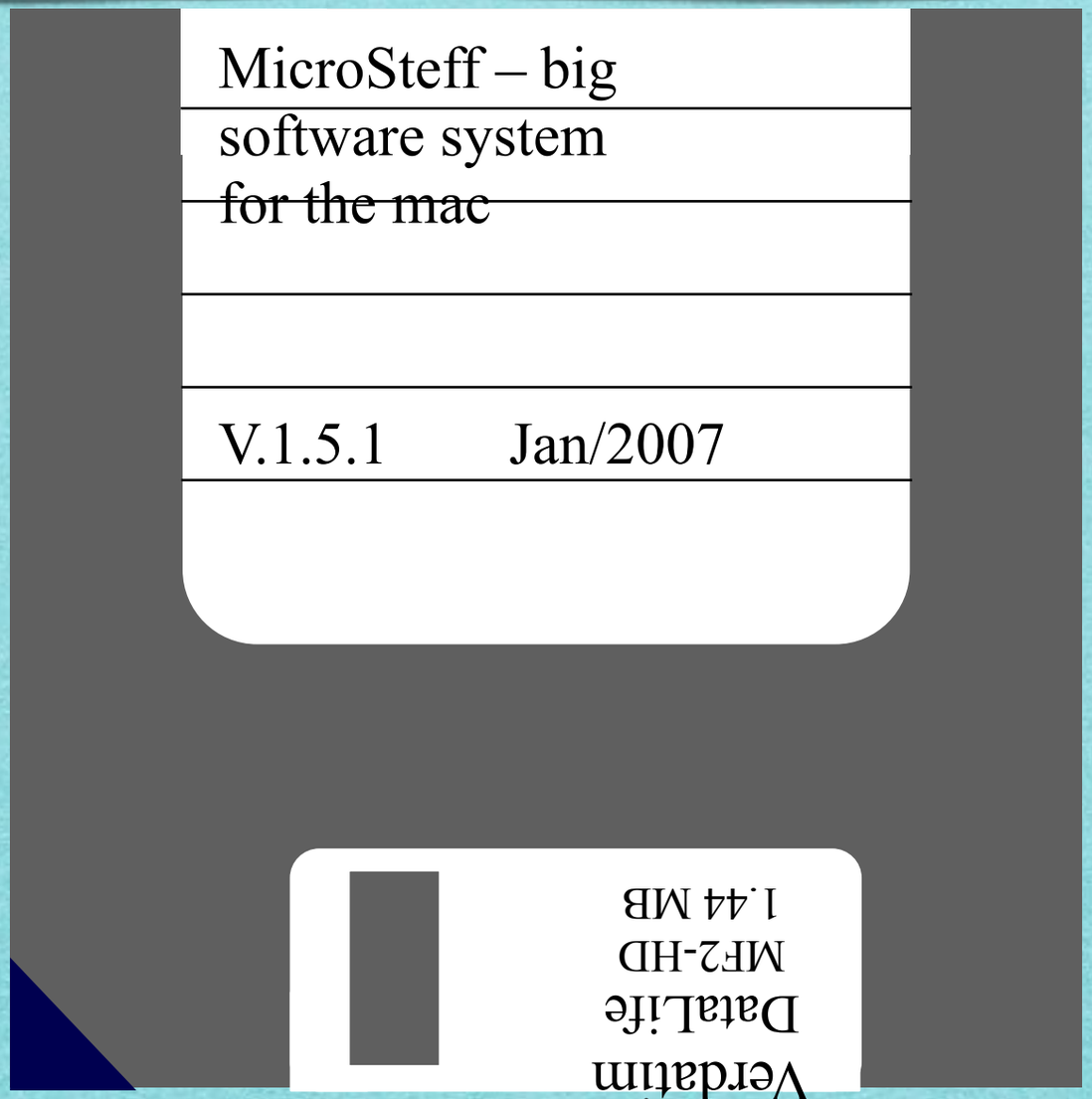
بیاید! تحویل شما!

اولین کار **حرفه‌ای** من!

یک پشته از برگه‌های پرینت شده! بدون هیچ راهنما یا فهرستی!

بیاید! تحویل شما!

اولین کار **حرفه‌ای** من!



یک پشته از برگه‌های پرینت شده! بدون هیچ راهنما یا فهرستی!

بیاید! تحویل شما!

اولین کار **حرفه‌ای** من!

برنامه بزرگ! (تو فلاپی جاش نشده!)

Jan/2007

یک پشته از برگه‌های پرینت شده! بدون هیچ راهنما یا فهرستی!

هزینه آزمون

- در دنیای واقعی، آزمون کاری است که پس از طراحی انجام می‌شود.
- عدم انجام آزمون‌های اولیه سبب افزایش هزینه می‌شود.
- درگیری زیاد نرم‌افزار و سخت‌افزار سبب سخت‌تر شدن و زیادتر شدن تعداد آزمون‌ها می‌شود.

هزینه آزمون

- در دنیای واقعی، آزمون کاری است که پس از طراحی انجام می‌شود.
- حداقل نصف بودجه پروژه را برای آزمون نگهداری کنید! چه بخواید و چه نخواید!
- درگیری زیاد نرم‌افزار و سخت‌افزار سبب سخت‌تر شدن و زیادتر شدن تعداد آزمون‌ها می‌شود.

قسمت اول: چرا آزمون؟

- اهداف و دلایل و نیازمندی‌های آزمون معمولا نوشته نمی‌شوند
- برنامه ریزی برای چه میزان پوشش آزمون است؟
- چقدر آزمون کفایت می‌کند؟
- هدف مشترک - خرج بودجه

قسمت اول: چرا آزمون؟

- اهداف و دلایل و نیازمندی‌های آزمون معمولا نوشته نمی‌شوند

اگه نمی‌دونید چرا آزمون رو انجام می‌دید! کلاهتون پس معرکه است و همون بهتر که انجام ندهید!

- چقدر آزمون کفایت می‌کند؟
- هدف مشترک - خرج بودجه

چرا آزمون؟

- سال ۱۹۸۰: نرم‌افزار باید قابل نگهداری باشد
- حد مشخص شده برای قابلیت اطمینان؟
- هر آزمون به دنبال اثبات چه حقیقتی است؟
- تیم تحلیل نیازمندی‌ها حتما نیاز به حضور آزمونگر دارد.

چرا آزمون؟

- سال ۱۹۸۰: نرم‌افزار باید قابل نگهداری باشد

اگر برای هر آزمون، در زمان تحلیل نیازمندی‌ها برنامه‌ای نداشته باشید، قطعا بعدا نمی‌دونید چرا دارید آزمون انجام می‌دین!

- تیم تحلیل نیازمندی‌ها حتما نیاز به حضور آزمونگر دارد.

هزینه عدم آزمون

- عدم آزمون ارزون نیست! حتی گرون‌تر از آزمون کردن هست!
- برنامه‌ریزی برای آزمون بعد از کد قطعا بسیار گرونه
- یه مکان برای آزمون بردهای الکترونیکی نیم میلیون دلار هزینه داره
- آزمون نرم‌افزاری همون برد ۱۰ هزار دلار هم هزینه نداره

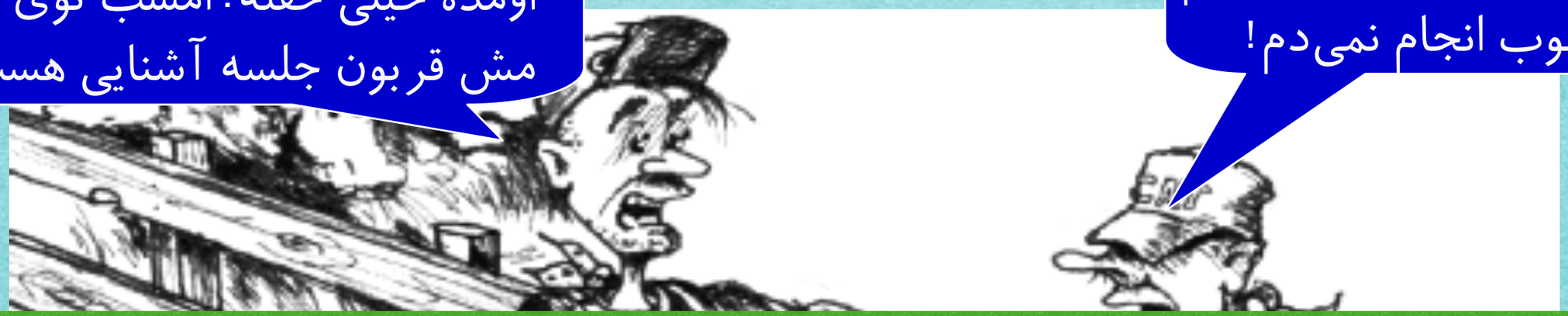
هزینه عدم آزمون

- عدم آزمون ارزون نیست! حتی گرون‌تر از آزمون کردن هست!
- برنامه‌ر [illegible] ونه
- یه مکا [illegible] ن دلار هزینه داره
- آزمون نرم‌افزاری همون برد ۱۰ هزار دلار هم هزینه نداره

مدیران معمولا می‌گن:
آزمون نرم‌افزاری خیلی گرونه

اخطار : تاثیر ابزارهای جدید

اخطار : تاثیر ابزارهای جدید



“Knowing is not enough, we must apply.
Willing is not enough, we must do.”
Goethe

بخش دوم: چه چیزی؟

1. نوع فعالیت‌های آزمون
2. واژگان مربوط به آزمون
3. تغییر مفاهیم آزمون
   - معیارهای پوشش آزمون
   - معیارهایی روی ساختار

بخش دوم: چه چیزی؟

1. نوع فعالیت‌های آزمون
2. واژگان مربوط به آزمون
3. تغییر [illegible]

خوب؟ باید چه کار کنیم الان؟

- معیارهای پوشش آزمون
- معیارهایی روی ساختار

انواع فعالیت‌های مربوط به آزمون

- فعالیت‌های مربوط به آزمون رو می‌شه توی چهار دسته قرار داد:

1. Test Design
2. Test Automation
3. Test Execution
4. Test Evaluation

1.a) Criteria-based

1.b) Human-based

- هر نوع آزمون نیاز به تخصص متفاوتی داره! دانش متفاوتی می‌خواد و تحصیلاتش هم متفاوته
- هیچ شرکت با مدیریت درست!، از یه آدم برای انجام چهار کار طراحی، پیاده‌سازی، مجتمع‌سازی و مدیریت پیکربندی استفاده نمی‌کنه

انواع فعالیت‌های مربوط به آزمون

- فعالیت‌های مربوط به آزمون رو می‌شه توی چهار دسته قرار داد:

1. Test Design

چرا سازمان‌ها هنوز از افراد مشابهی برای انجام چهار نوع عمل استفاده می‌کنن؟

4. Test Evaluation

- هر نوع آزمون نیاز به تخصص متفاوتی داره! دانش متفاوتی می‌خواد و تحصیلاتش هم متفاوته
- هیچ شرکت با مدیریت درست!، از یه آدم برای انجام چهار کار طراحی، پیاده‌سازی، مجتمع‌سازی و مدیریت پیکربندی استفاده نمی‌کنه

انواع فعالیت‌های مربوط به آزمون

- فعالیت‌های مربوط به آزمون رو می‌شه توی چهار دسته قرار داد:

1. Test Design

چرا سازمان‌ها هنوز از افراد مشابهی برای انجام چهار نوع عمل استفاده می‌کنن؟

4. Test Evaluation

قطعا این کار هدر دادن وقته! اما چرا؟

- هر نوع ... هم متفاوته
- هیچ شرکت با مدیریت درست!، از یه آدم برای انجام چهار کار طراحی، پیاده‌سازی، مجتمع‌سازی و مدیریت پیکربندی استفاده نمی‌کنه

1. Test Design – (a) Criteria-Based

1. Test Design – (a) Criteria-Based

- تکنیکی‌ترین کار در انجام آزمون طراحی آزمون است
- نیاز به دانش
  - ریاضیات گسسته
  - برنامه‌نویسی
  - آزمون
- نیاز به مدرک علوم کامپیوتر دارد
- کار هیجان‌انگیز، سخت و چالشی است
- یه چیزی مثل طراحی معماری در رشته‌های دیگر
- استفاده از افراد ناشی برای طراحی آزمون سبب پشیمانی بزرگ و آزمون‌های بی‌خاصیت خواهد شد!

1. Test Design – (a) Criteria-Based

- تکنیکی‌ترین کار در انجام آزمون طراحی آزمون است
- نیاز به دانش
  - ریاضیات گسسته
  - برنامه‌نویسی
- کار هیجان‌انگیز، سخت و چالشی است
- یه چیزی مثل طراحی معماری در رشته‌های دیگر
- استفاده از افراد ناشی برای طراحی آزمون سبب پشیمانی بزرگ و آزمون‌های بی‌خاصیت خواهد شد!

طراحی مقادیر آزمون برای پوشش معیارهای آزمون و یا اهداف دیگر مهندسی

1. Test Design – (b) Human-Based

- این مورد از چیزی که به نظر می‌رسد دشوارتر است!
- روش مبتنی بر معیار در برخی موارد کور است
- نیاز به دانش مربوط به دامنه، آزمون و واسط کاربری است
- معمولا نیاز به دانش علوم کامپیوتر سابق ندارد
  - یک سابقه از دامنه مهندسی مورد نیاز است
  - دانش تجربی از دامنه مورد توجه مانند بیولوژی، روانشناسی و ...
  - پیش زمینه منطقی مانند ریاضی، فلسفه و ...
- این روش از لحاظ فکری جالب، چالشی و سخت است
- اما نه به اندازه روش مبتنی بر معیار!

1. Test Design – (b) Human-Based

- این مورد از چیزی که به نظر می‌رسد دشوارتر است!
- روش مبتنی بر معیار در برخی موارد کور است
- نیاز به دانش مربوط به دامنه، آزمون و واسط کاربری است

طراحی مقادیر آزمون بر اساس دانش مربوط به نرم‌افزار و دانش تجربی افراد

- دانش تجربی از دامنه مورد توجه مانند بیولوژی، روانشناسی و ...
- پیش زمینه منطقی مانند ریاضی، فلسفه و ...
- این روش از لحاظ فکری جالب، چالشی و سخت است
- اما نه به اندازه روش مبتنی بر معیار!

2. Test Automation

2. Test Automation

- این مورد کمتر تکنیکی است
- نیاز به دانش مربوط به برنامه‌نویسی دارد
  - برنامه نویسی معمولا سر راست و ساده است.
- نیاز به تئوری‌های محدود دارد.
- برای طراحان خسته کننده است.
- چه کسی مسئول تشخیص و قرار دادن مقادیر مورد انتظار در خروجی نرم‌افزار است؟
  - طراحان آزمون معمولا مقادیر خروجی را نمی‌دانند.
  - ارزیابان آزمون برای کمک به شناخت مقادیر خروجی باید درگیر این قدم شوند.

2. Test Automation

- این مورد کمتر تکنیکی است
- نیاز به دانش مربوط به برنامه‌نویسی دارد
  - برنامه نویسی معمولا سر راست و ساده است.
- نیاز به تئوری‌های محدود دارد.

مقادیر آزمون را در اسکریپت‌های اجرایی قرار دهید.

- چه کسی مسئول تشخیص و قرار دادن مقادیر مورد انتظار در خروجی نرم‌افزار است؟
  - طراحان آزمون معمولا مقادیر خروجی را نمی‌دانند.
  - ارزیابان آزمون برای کمک به شناخت مقادیر خروجی باید درگیر این قدم شوند.

3. Test Execution

3. Test Execution

- کاری بسیار ساده (البته اگر خودکارسازی آزمون درست انجام شده باشد)
- نیاز به مهارت‌های کامپیوتری پایه
  - کارآموزان
  - کارمندان بدون پیش‌زمینه تکنیکی
- استفاده از طراحان آزمون برای اجرای آزمون! یک پیشنهاد قطعی به جهت تغییر شغل آن‌ها حساب می‌شود!
- اگر برخی بخش‌ها به درستی خودکارسازی نشده باشند، نیاز به کارهای دستی زیادی است. مثل آزمون GUI
- اجرا کنندگان آزمون در ثبت نتایج باید بسیار مواظب و دقیق باشند

3. Test Execution

- کاری بسیار ساده (البته اگر خودکارسازی آزمون درست انجام شده باشد)
- نیاز به مهارت‌های کامپیوتری پایه
  - کارآموزان

اجرای آزمون روی نرم‌افزار و ثبت نتایج

- استفاده از طراحان ازمون برای اجرای ازمون! یک پیشنهاد قطعی به جهت تغییر شغل آن‌ها حساب می‌شود!
- اگر برخی بخش‌ها به درستی خودکارسازی نشده باشند، نیاز به کارهای دستی زیادی است. مثل آزمون GUI
- اجرا کنندگان آزمون در ثبت نتایج باید بسیار مواظب و دقیق باشند

4. Test Evaluation

4. Test Evaluation

- این مورد بسیار سخت‌تر از آن چیزی است که به نظر می‌رسد
- نیاز به دانش دامنه، آزمون، واسط کاربر و روانشناسی دارد
- معمولا نیازی به دانش سنتی علوم کامپیوتر ندارد
  - یک پیش زمینه از دامنه
  - دانش تجربی بسیار مفید است
  - یک پیش زمینه منطقی مفید است
- این مورد کاری جذاب، هیجانی و چالش بر انگیز است.

4. Test Evaluation

- این مورد بسیار سخت‌تر از آن چیزی است که به نظر می‌رسد
- نیاز به دانش دامنه، آزمون، واسط کاربر و روانشناسی دارد
- معمولا نیازی به دانش سنتی علوم کامپیوتر ندارد

ارزیابی نتایج اجرای آزمون و گزارش به توسعه دهنده‌گان

- دانش تجربی بسیار مفید است
- یک پیش زمینه منطقی مفید است
- این مورد کاری جذاب، هیجانی و چالش بر انگیز است.

سایر فعالیت‌ها

- **مدیریت آزمون**: قرار دادن سیاست‌ها، سازمان دهی تیم‌ها، ارتباط با توسعه‌دهنده‌گان، انتخاب معیار
- **نگهداری آزمون**: آزمون باید قابل استفاده مجدد باشد.
  - نیاز به همکاری طراحان و خودکارسازان تست
  - تصمیم‌گیری در مورد کم یا زیاد کردن آزمون‌ها معمولا سیاسی است
- **مستندسازی آزمون**:
  - هر مورد آزمون باید مستند شود. دلیل آزمون و خروجی‌های مورد انتظار و دلایل ایجاد آزمون
  - قابلیت ردگیری باید وجود داشته باشد
  - مستندسازی در اسکریپت آزمون‌ها نیز باید وجود داشته باشد

تخمینی از میزان پرسنل مورد نیاز

- در یک بخش تست بالغ، یک طراح آزمون با تعداد زیادی خودکارساز، مجری و ارزیاب آزمون فعالیت می‌کند
- خودکارسازی مناسب آزمون، تعداد مجری‌های آزمون را کمتر می‌کند
  - در حالت ایده‌آل صفر!
- قرار دادن آدم‌ها در جای نامناسب سبب عدم رضایت شغلی و پایین آمدن کارایی می‌شود
  - یک طراح آزمون با کارهای مربوط به اجرای آزمون خسته می‌شود.
  - یک ارزیاب آزمون معمولا توانایی درک اهمیت معیارها را ندارد.
- ارزیاب آزمون معمولا دانش دامنه را دارند، بنابراین می‌توانند آزمون‌هایی را اضافه کنند که آزمونگرهای عادی نمی‌توانند به آن‌ها فکر کنند.

انواع فعالیت‌ها ـ خلاصه

1a.	Design	Design test values to satisfy engineering goals
	Criteria	Requires knowledge of discrete math, programming and testing
1b.	Design	Design test values from domain knowledge and intuition
	Human	Requires knowledge of domain, UI, testing
2.	Automation	Embed test values into executable scripts
		Requires knowledge of scripting
3.	Execution	Run tests on the software and record the results
		Requires very little knowledge
4.	Evaluation	Evaluate results of testing, report to developers
		Requires domain knowledge

- این چهار فعالیت کاملا با هم متفاوت هستند
- طبیعتا استفاده از افراد یکسان برای هر ۴ فعالیت عاقلانه نیست

انواع فعالیت‌ها ـ خلاصه

1a.	Design	Design test values to satisfy engineering goals
	Criteria	Requires knowledge of discrete math, programming and testing
1b.	Design	Design test values from domain knowledge and intuition
	Human	Requires knowledge of domain, UI, testing
		Requires very little knowledge
4.	Evaluation	Evaluate results of testing, report to developers
		Requires domain knowledge

برخی از تیم‌های آزمون از افراد یکسانی برای تمامی فعالیت‌ها استفاده می‌کنند.

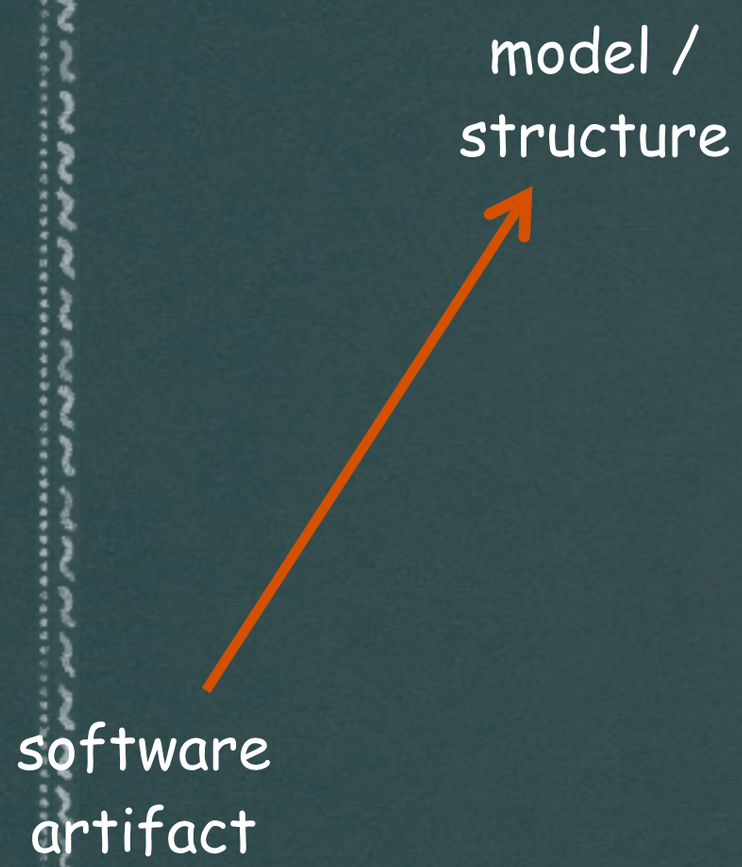
- این چهار فعالیت کاملا با هم متفاوت هستند
- طبیعتا استفاده از افراد یکسان برای هر ۴ فعالیت عاقلانه نیست

Model-Driven Test Design

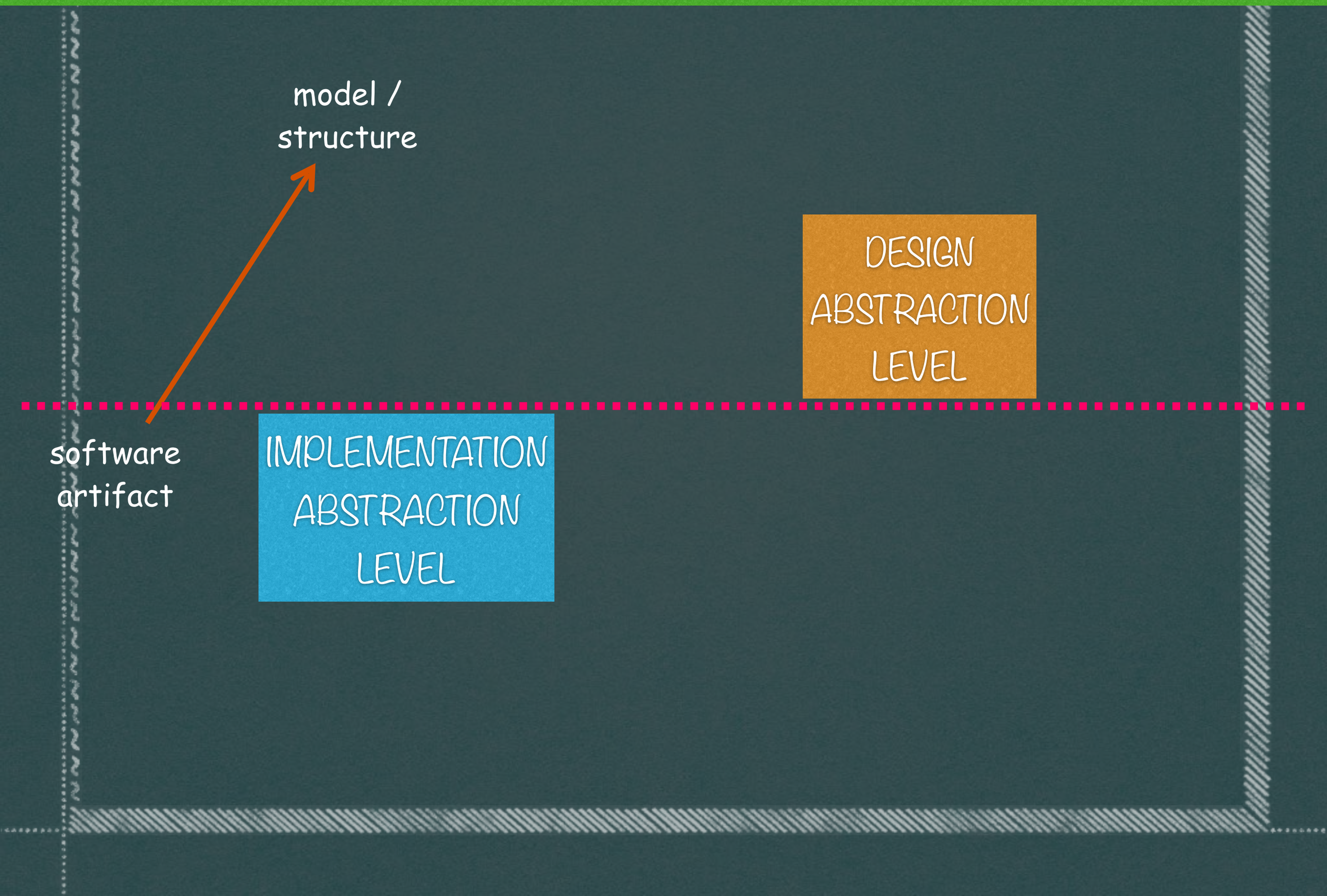
Model-Driven Test Design

software
artifact

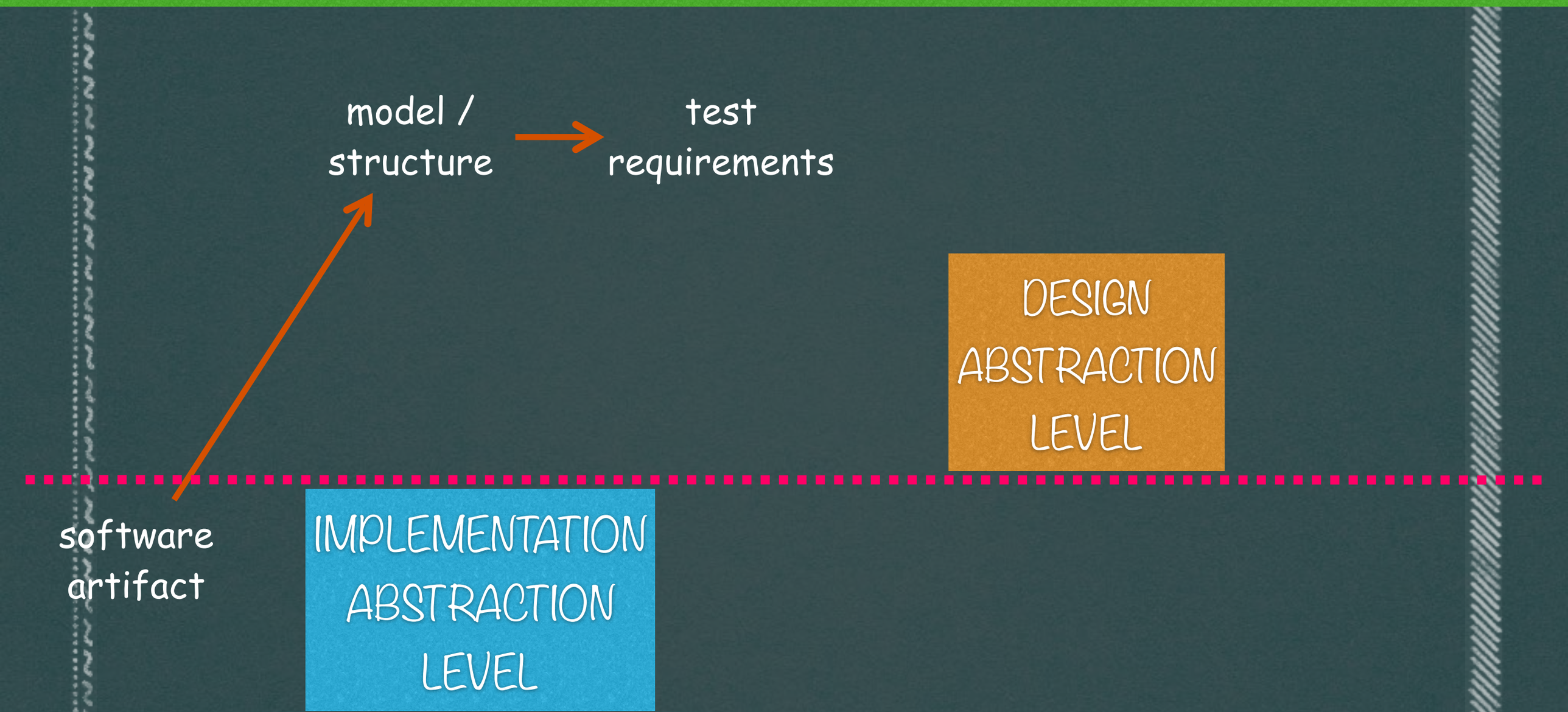
Model-Driven Test Design



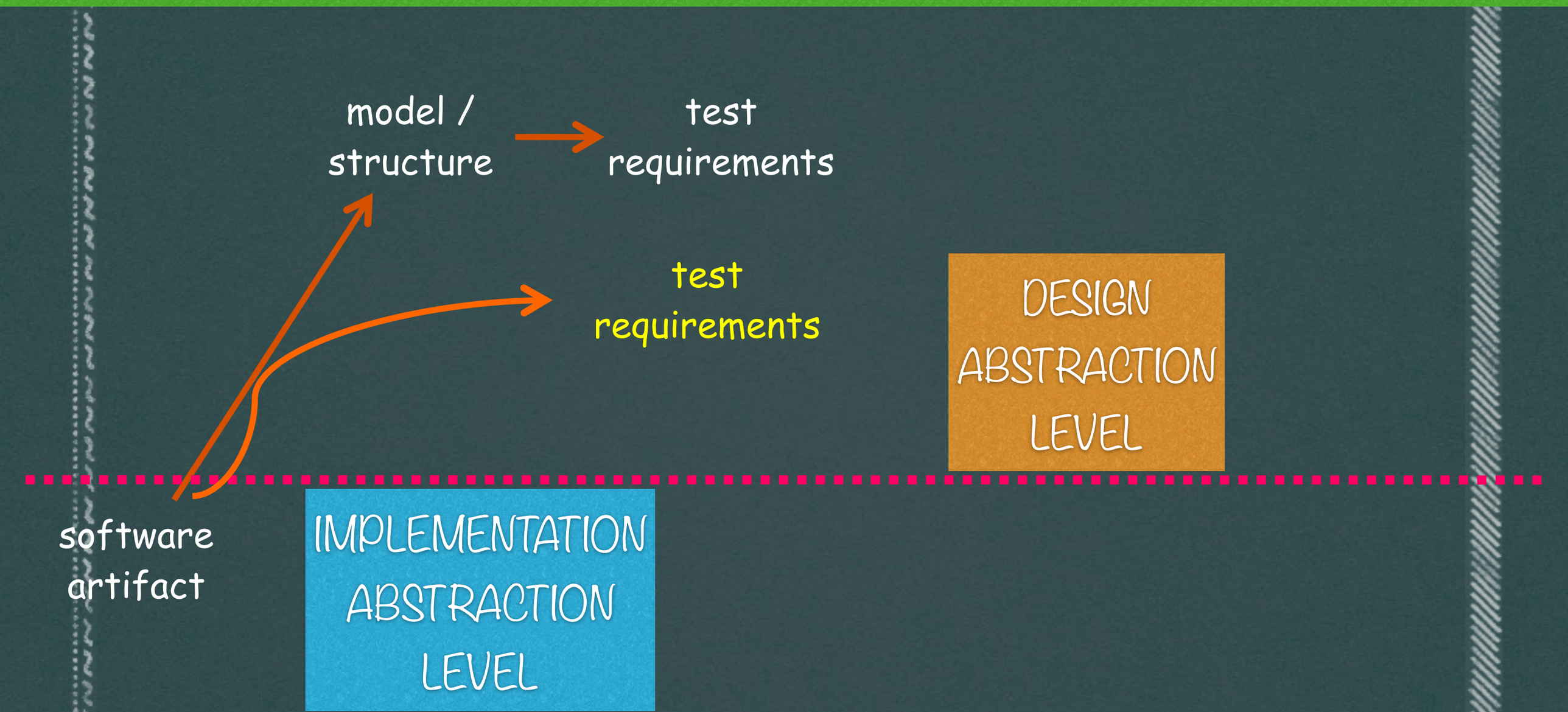
Model-Driven Test Design



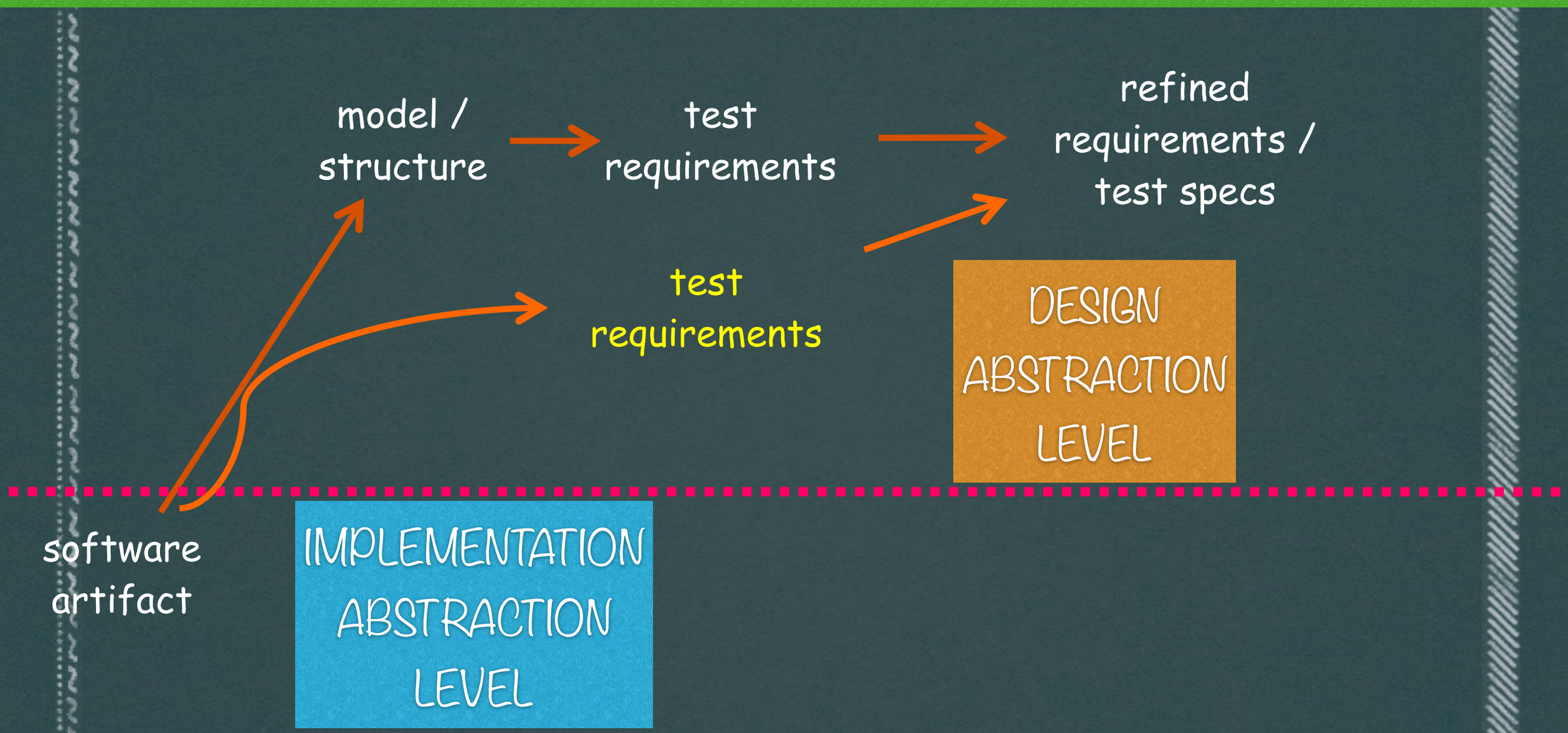
Model-Driven Test Design



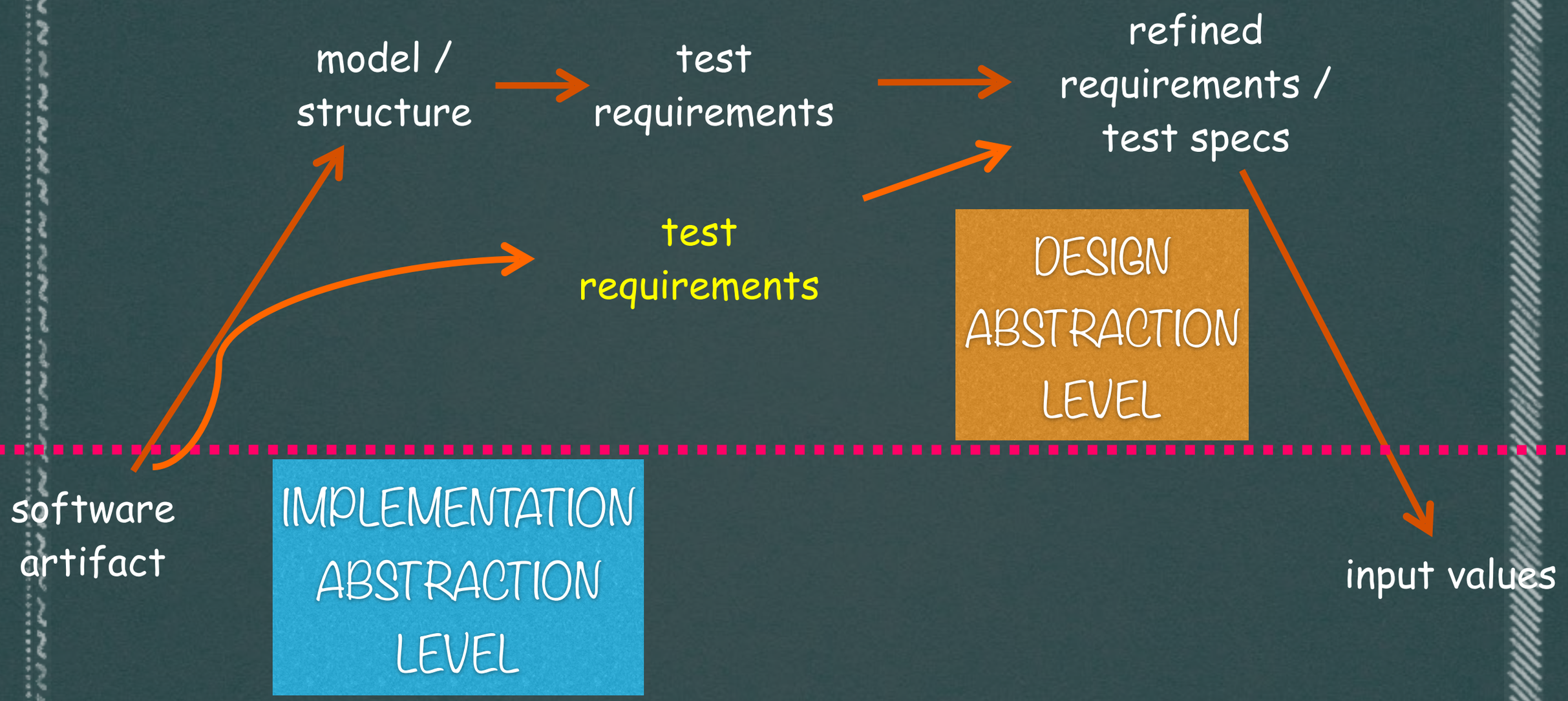
Model-Driven Test Design



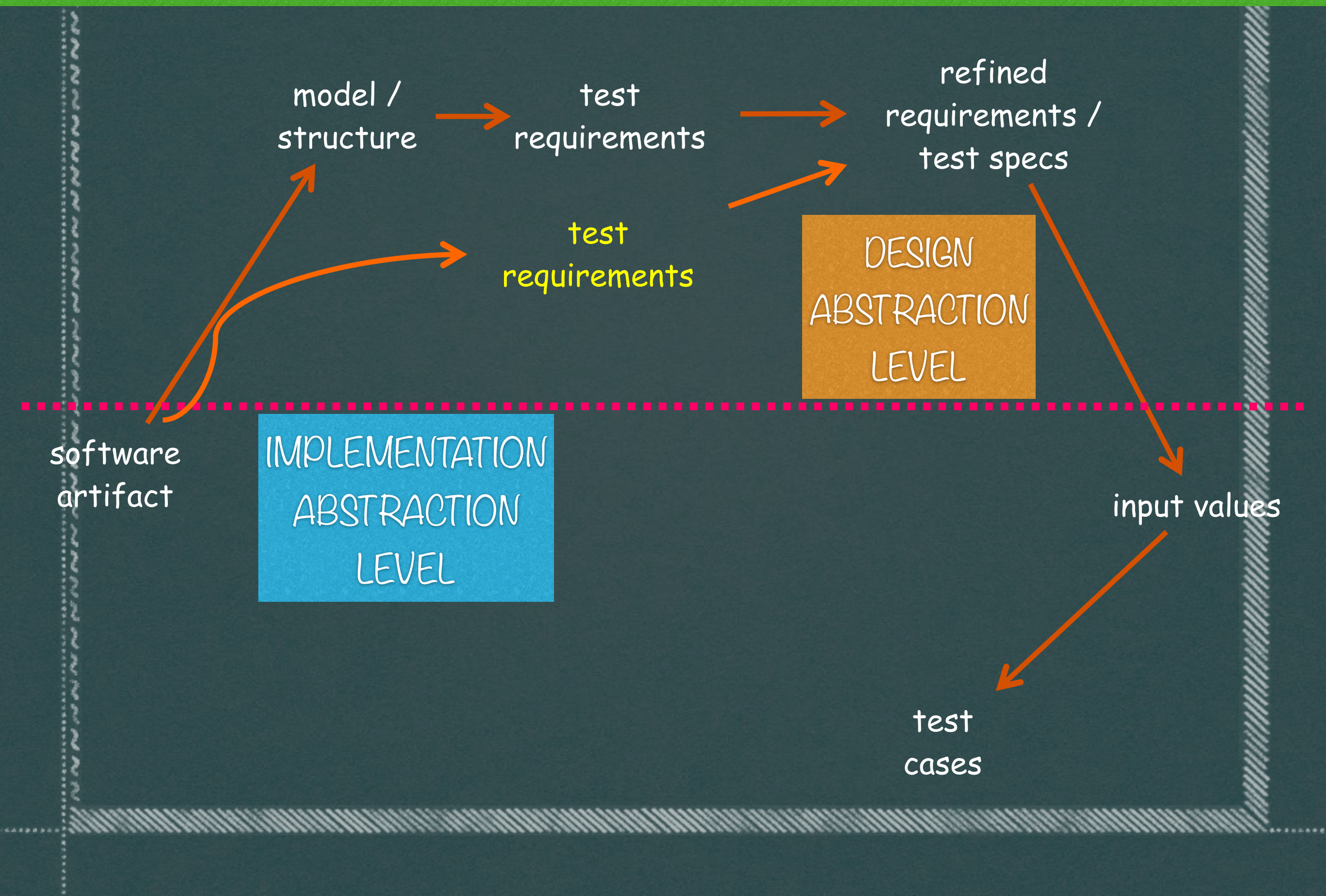
Model-Driven Test Design



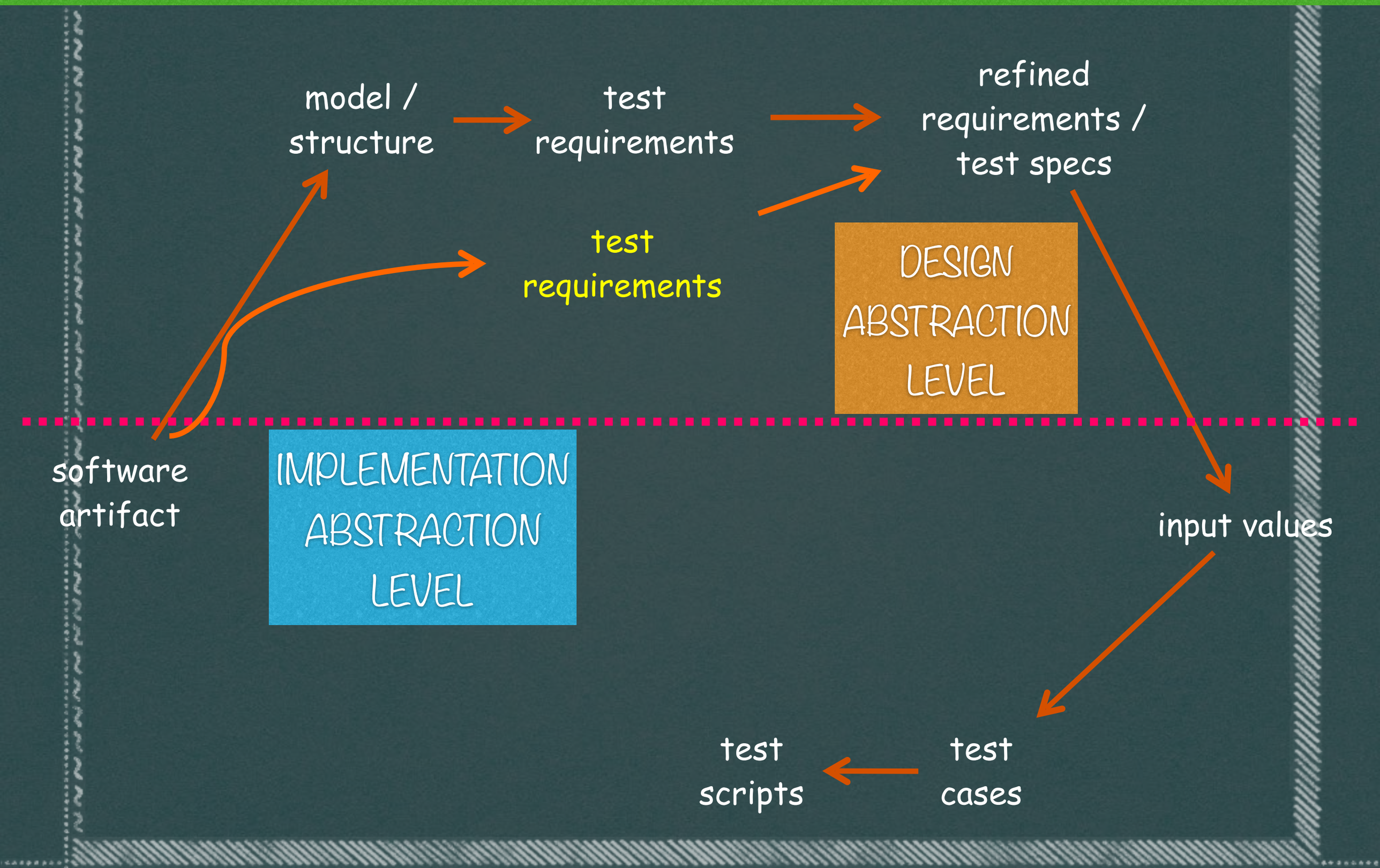
Model-Driven Test Design



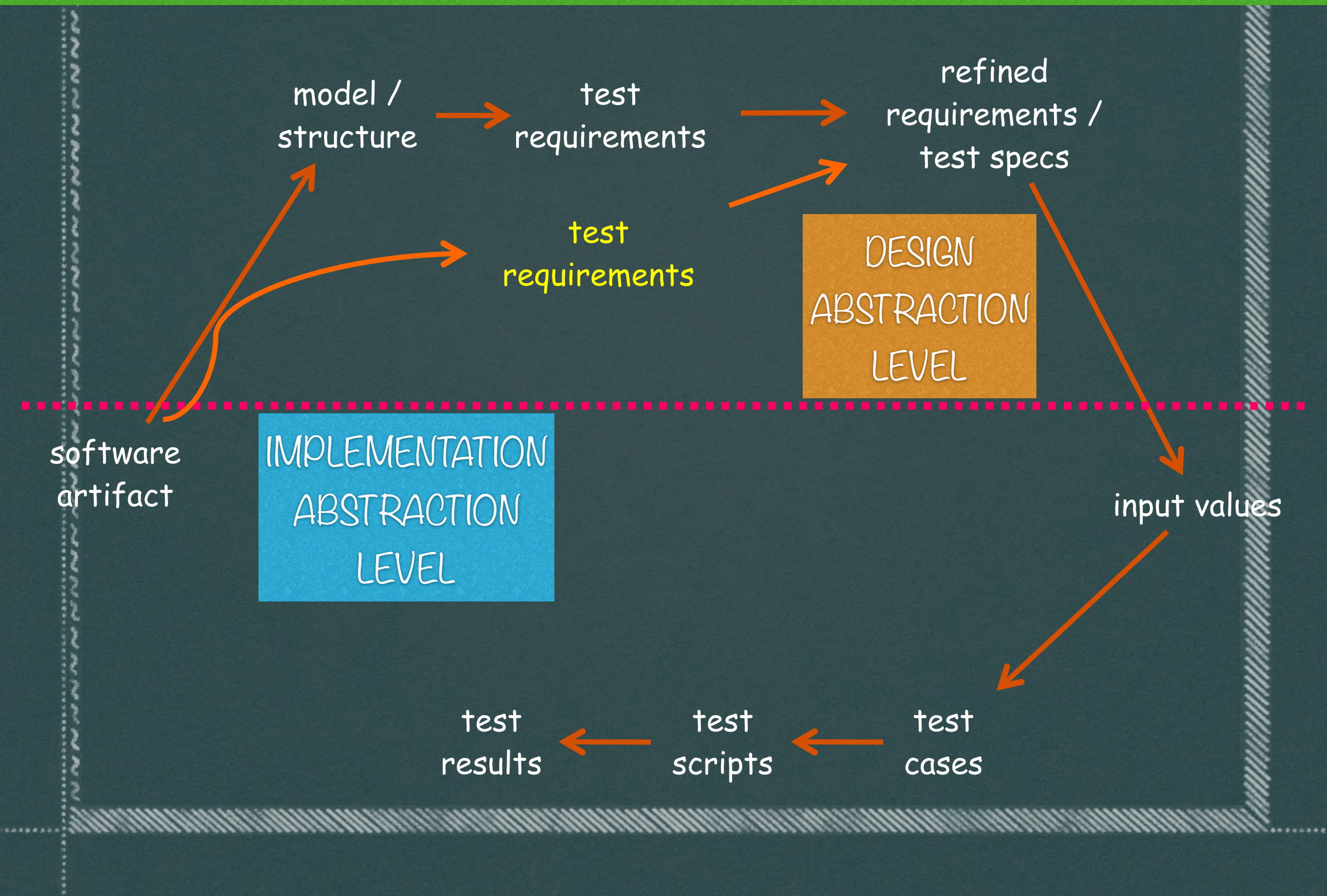
Model-Driven Test Design



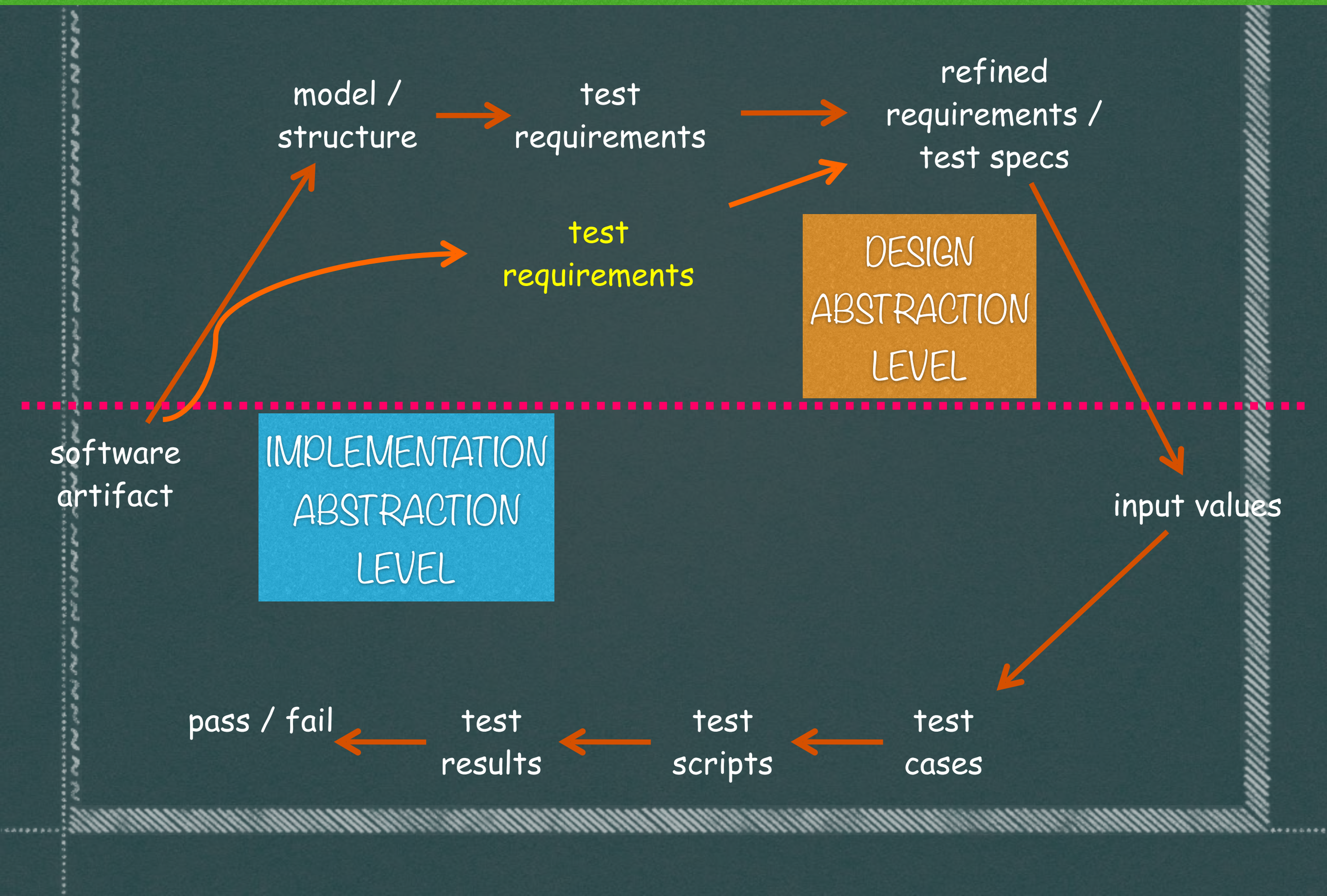
Model-Driven Test Design



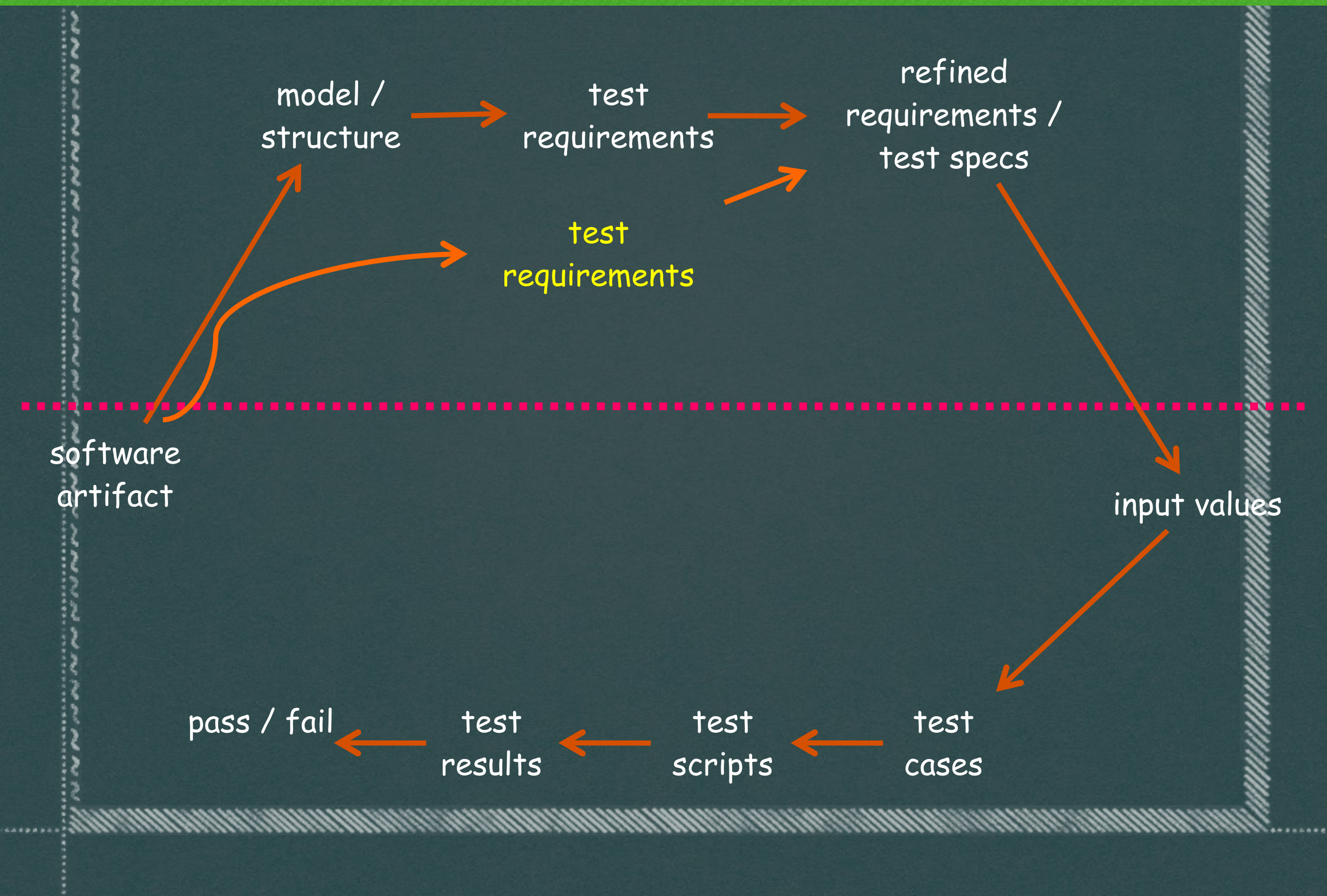
Model-Driven Test Design



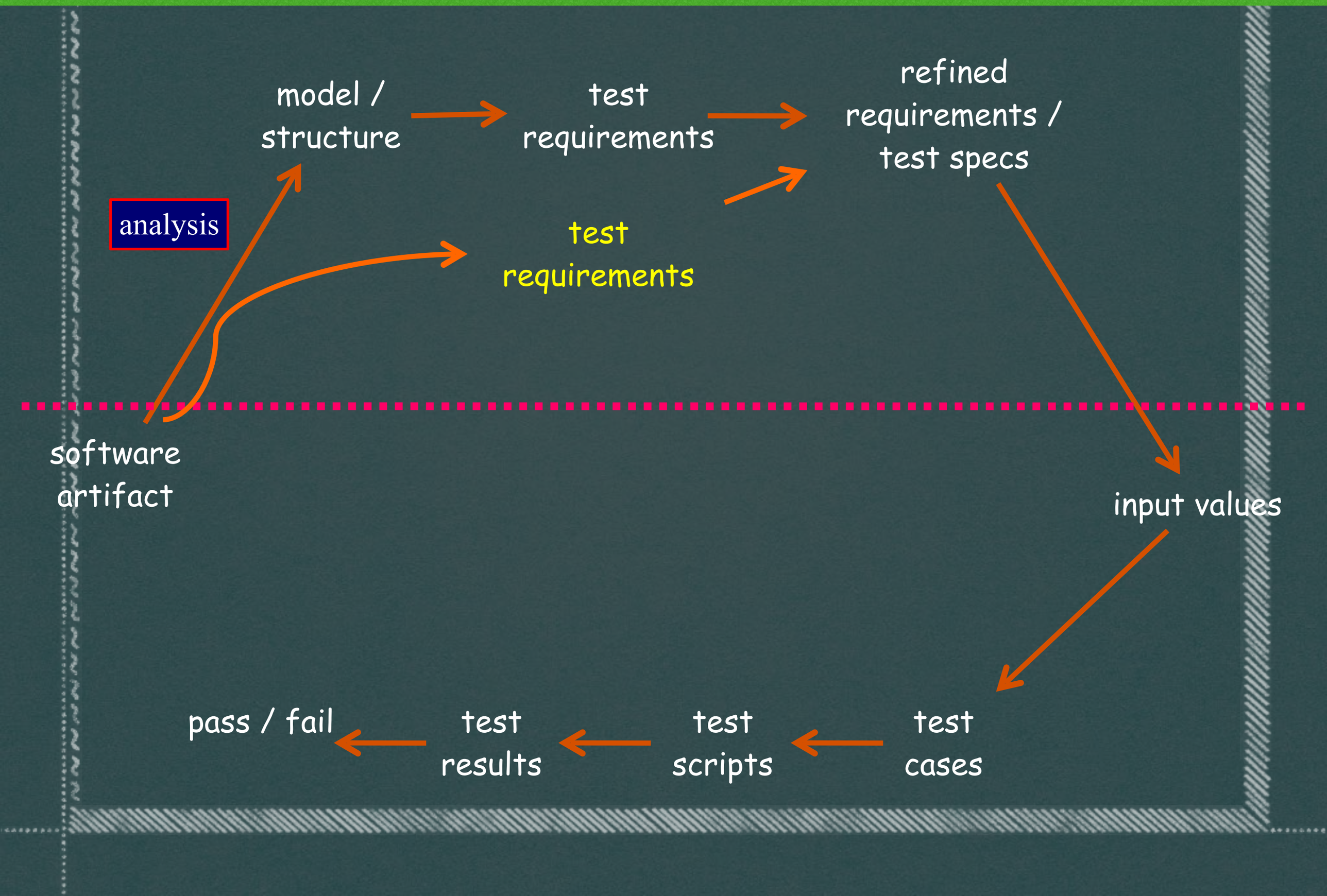
Model-Driven Test Design



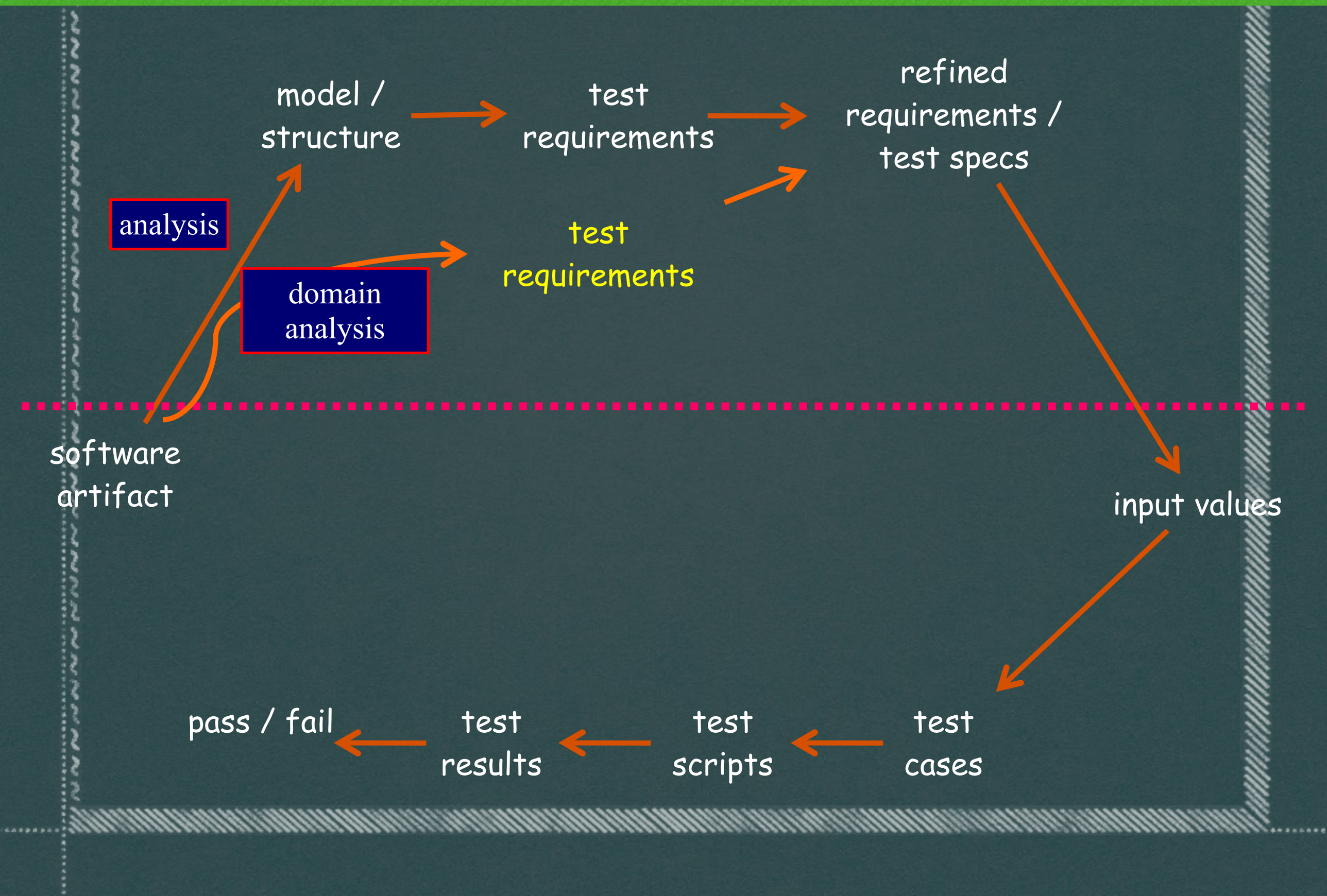
Model-Driven Test Design - Steps



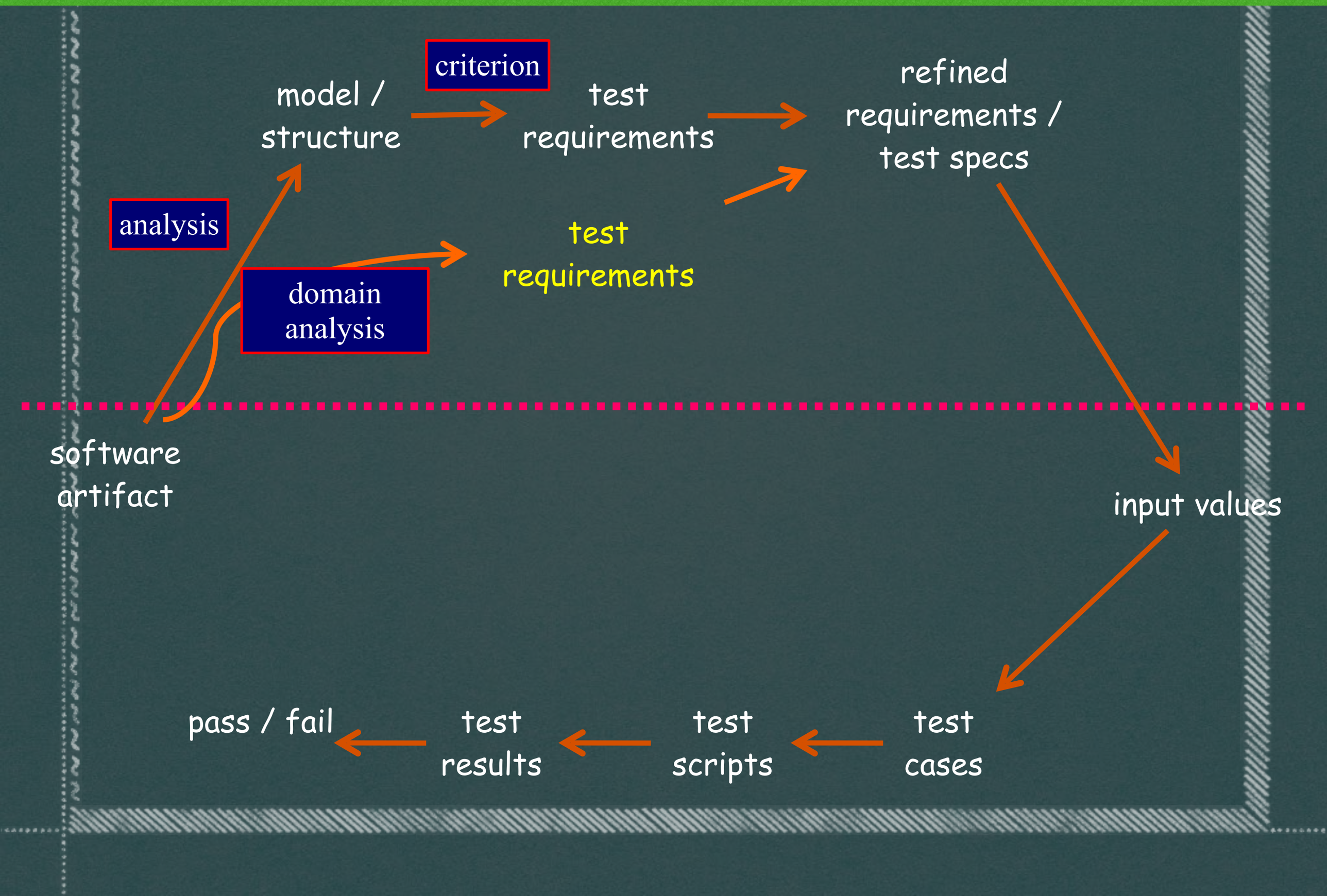
Model-Driven Test Design - Steps



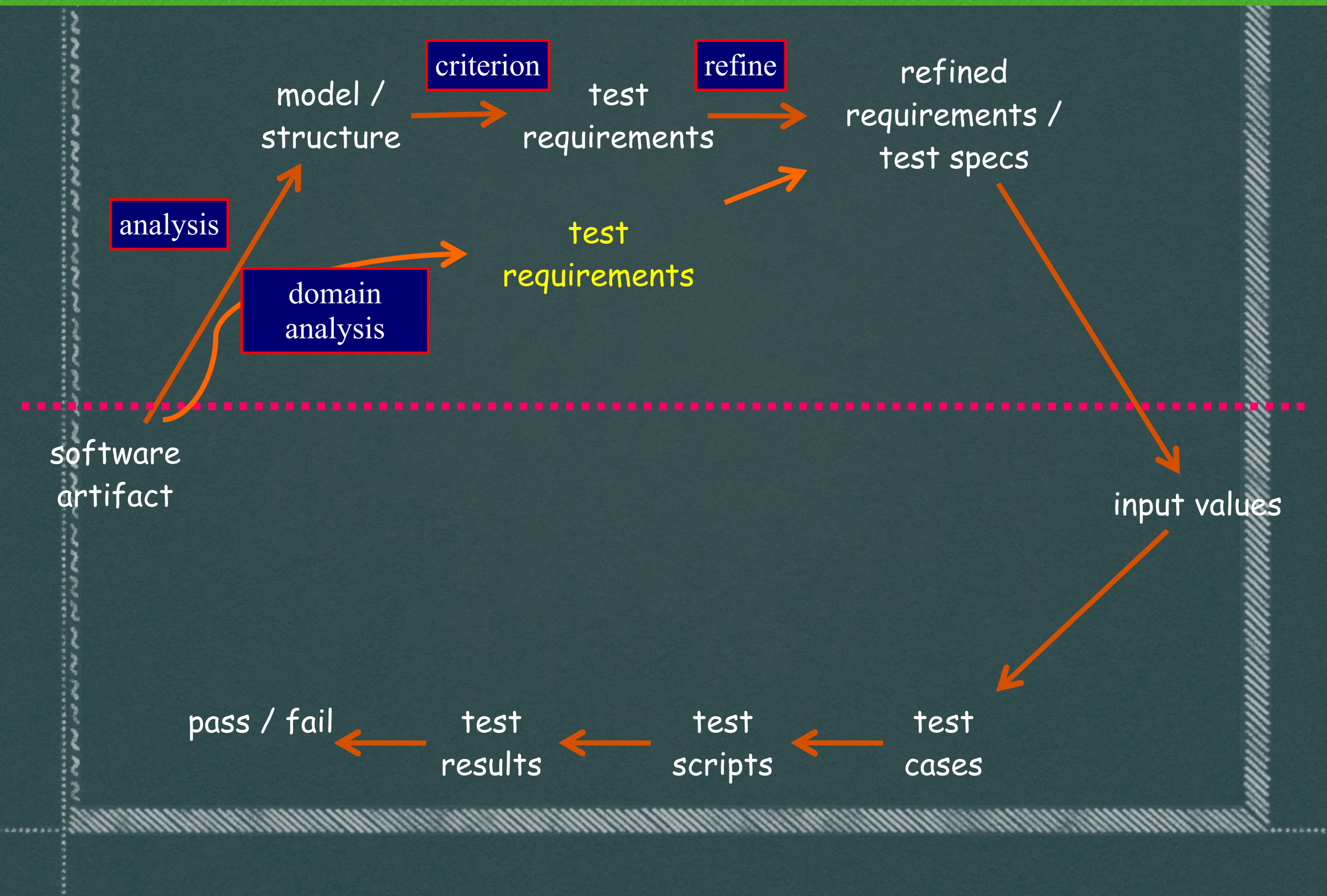
Model-Driven Test Design – Steps



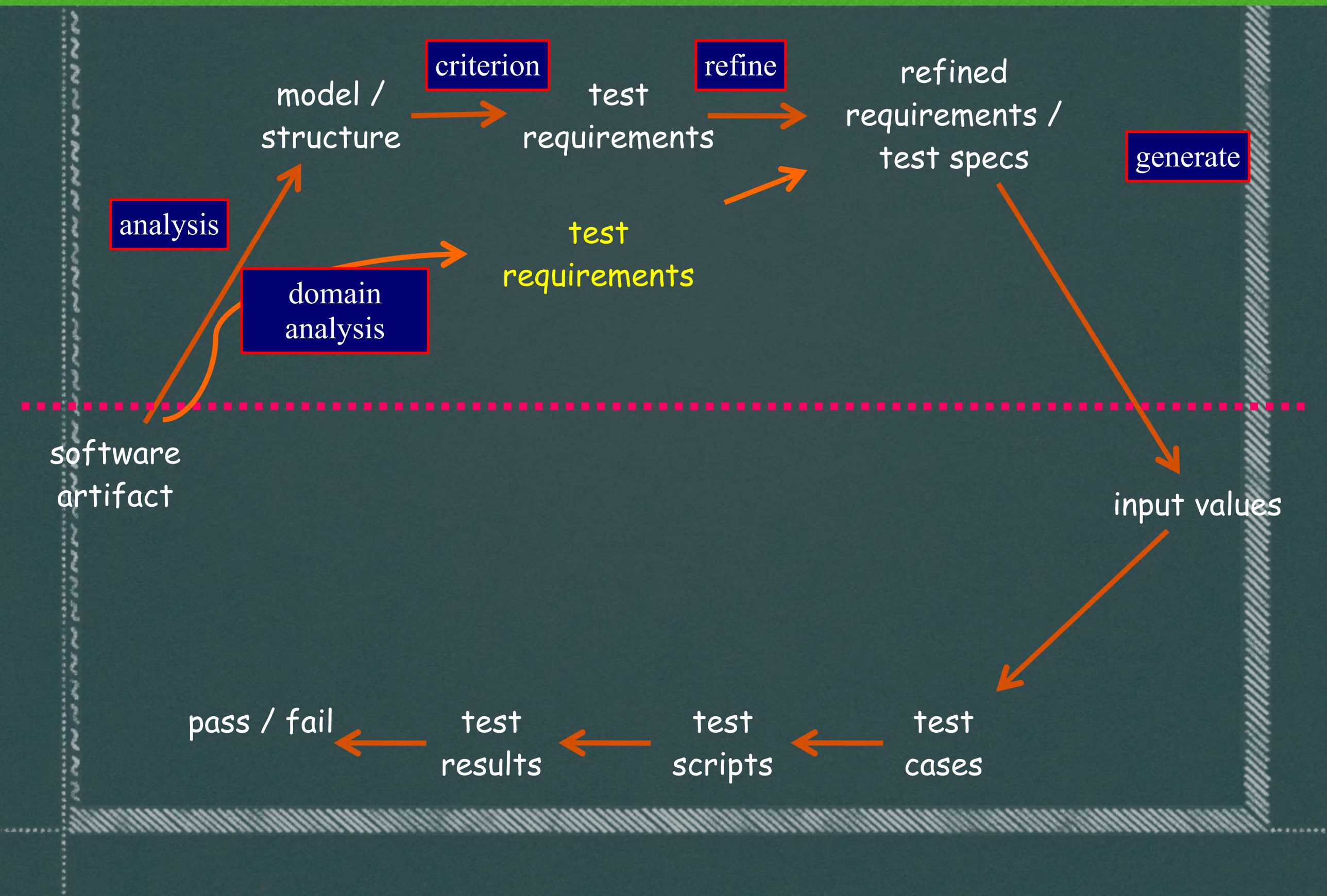
Model-Driven Test Design – Steps



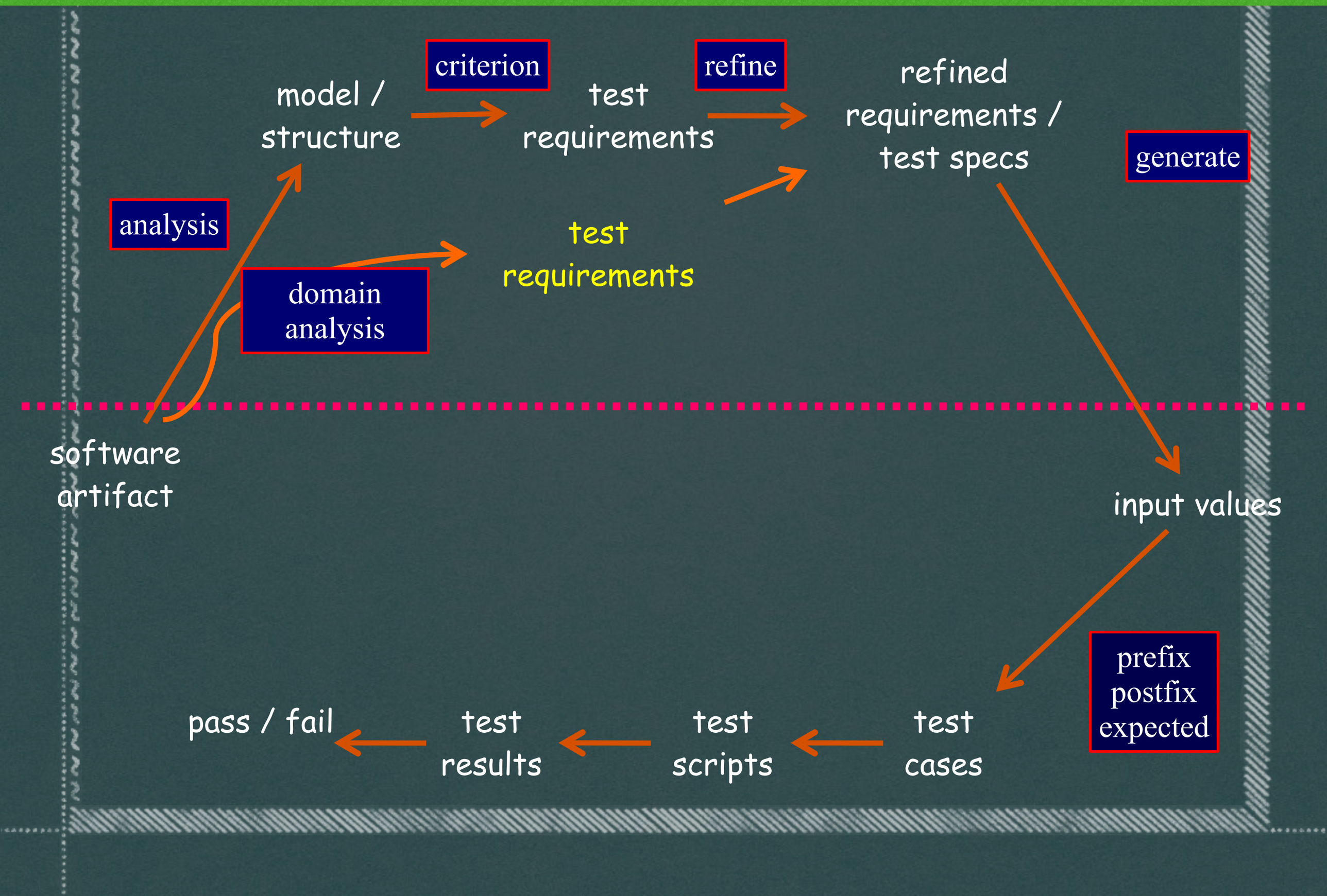
Model-Driven Test Design – Steps



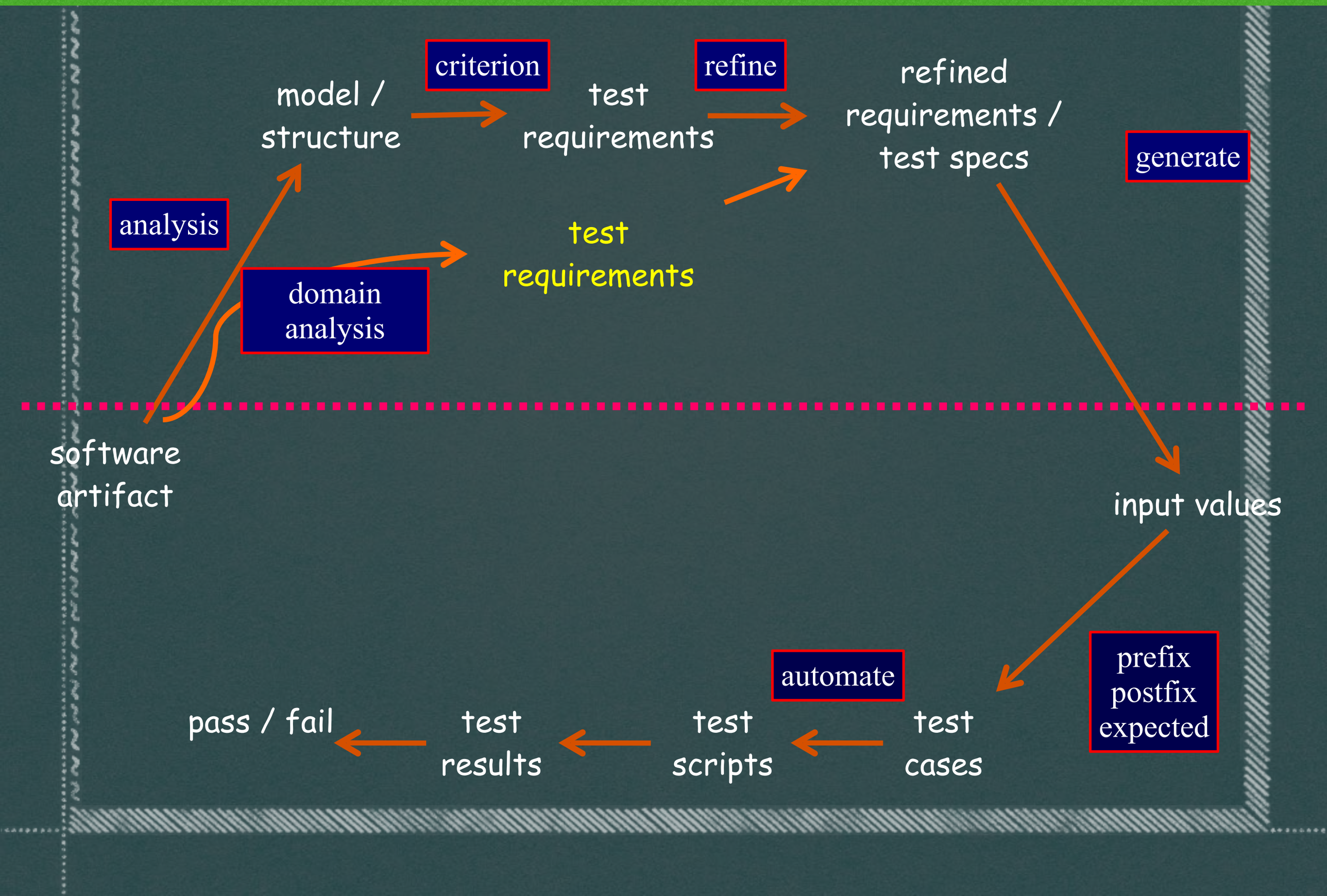
Model-Driven Test Design – Steps



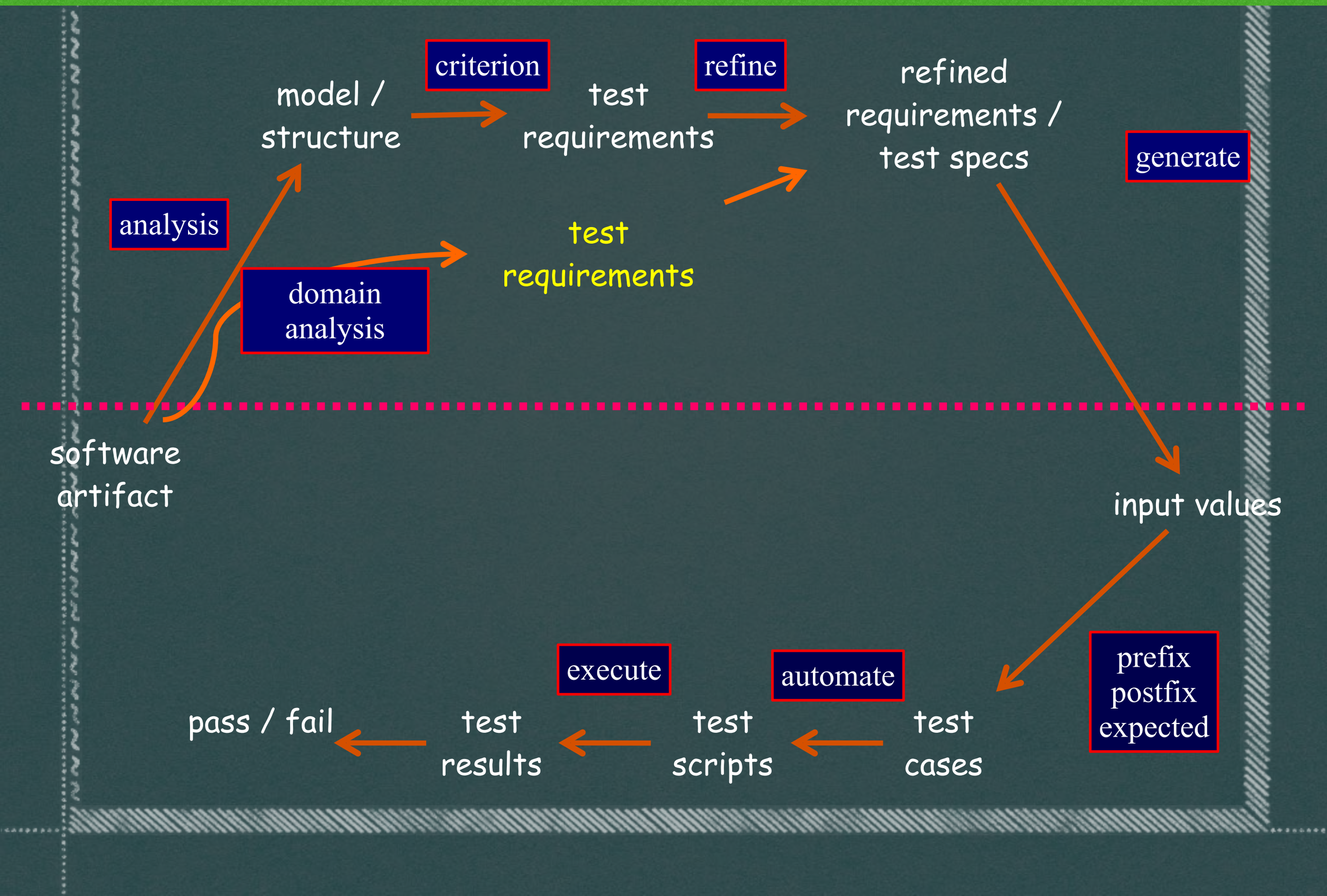
Model-Driven Test Design – Steps



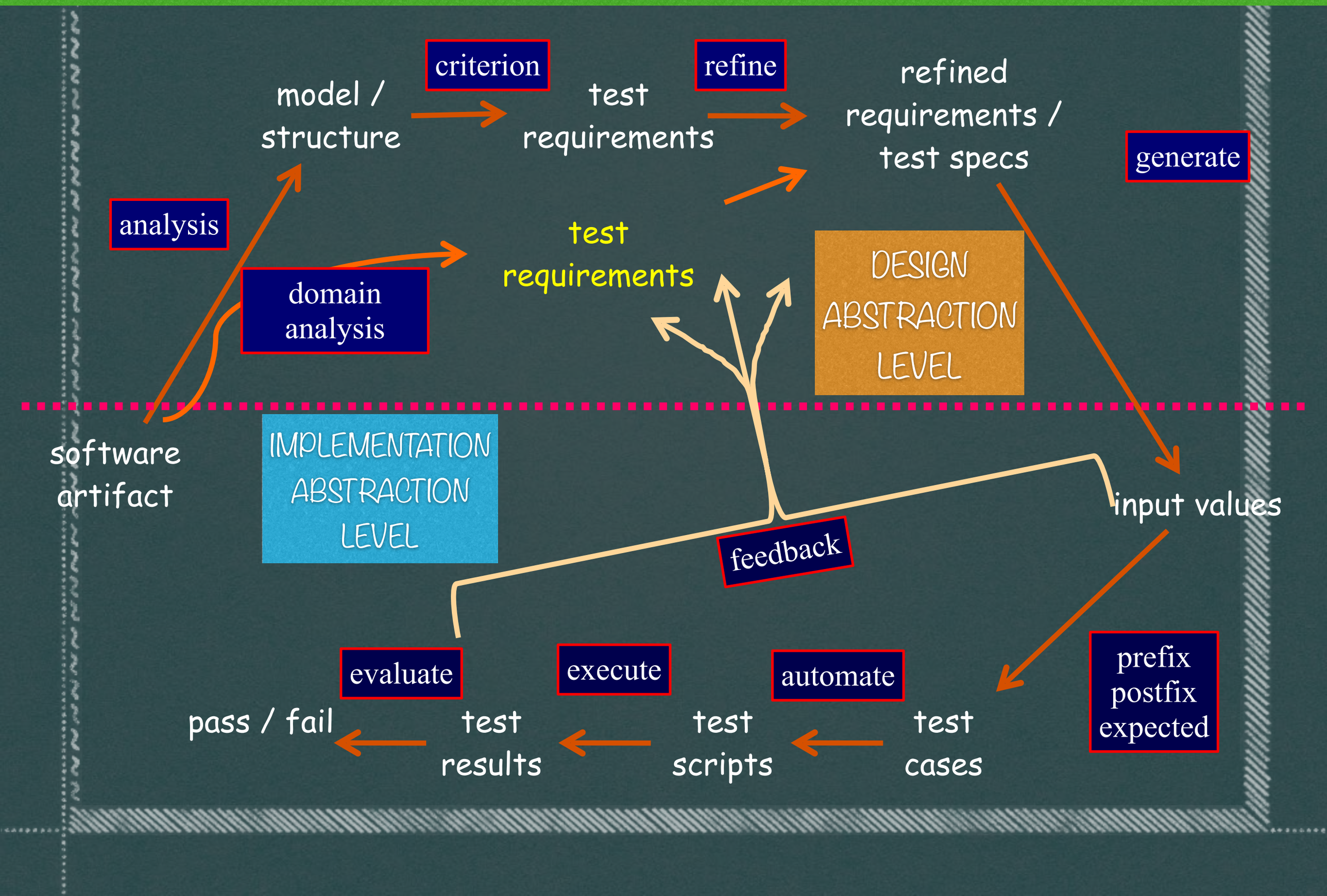
Model-Driven Test Design – Steps



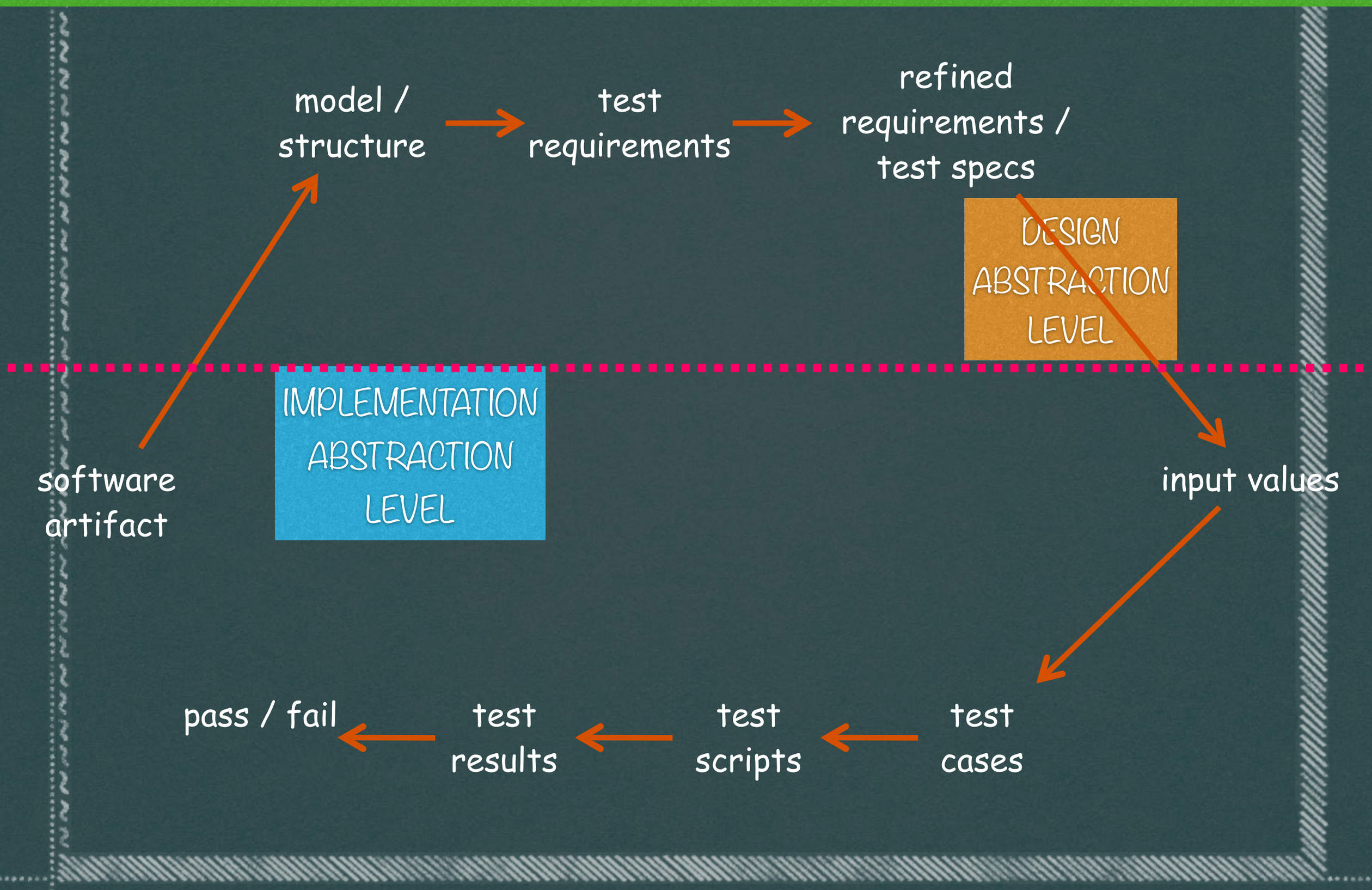
Model-Driven Test Design – Steps



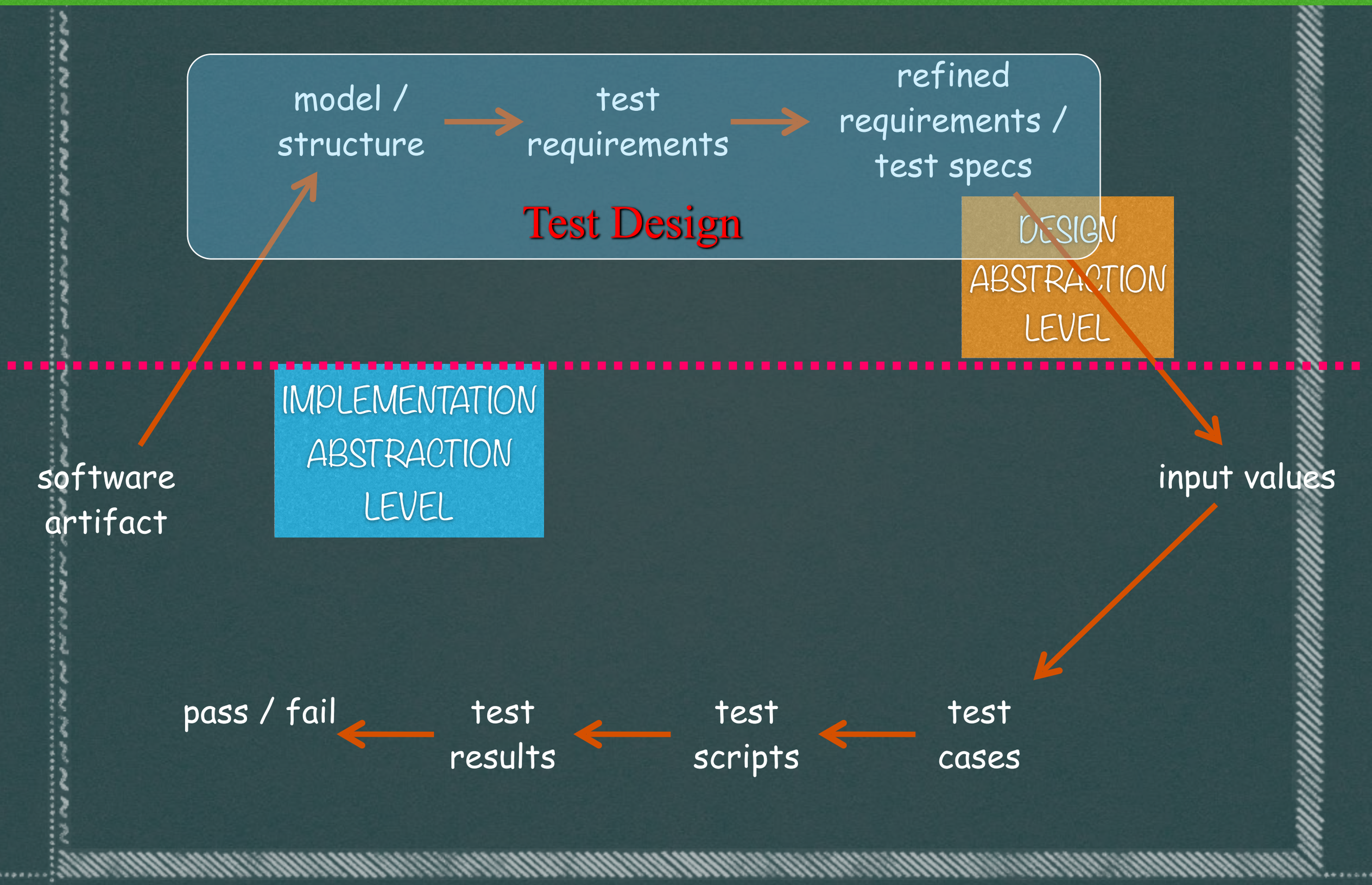
Model-Driven Test Design - Steps



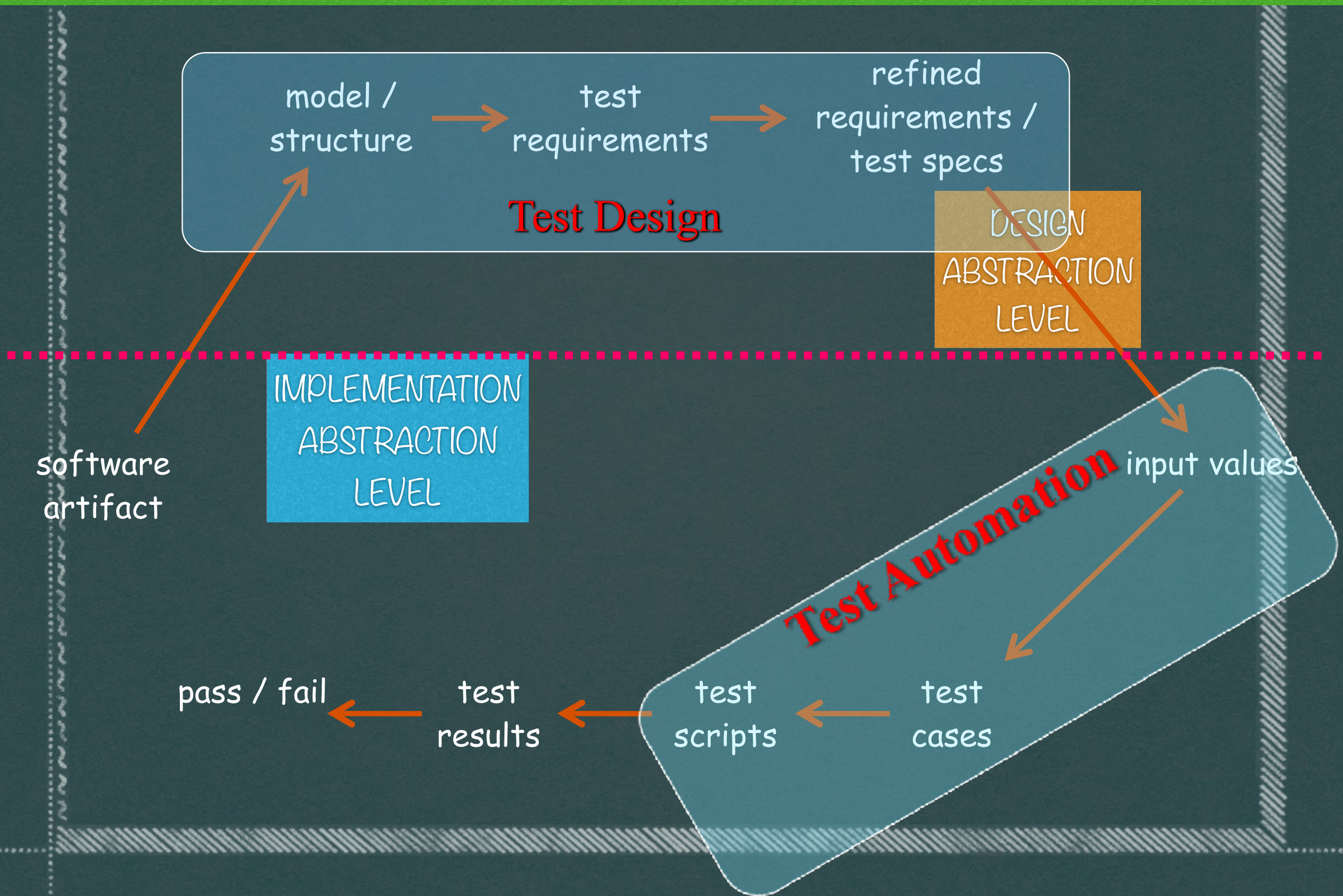
Model-Driven Test Design - Activities



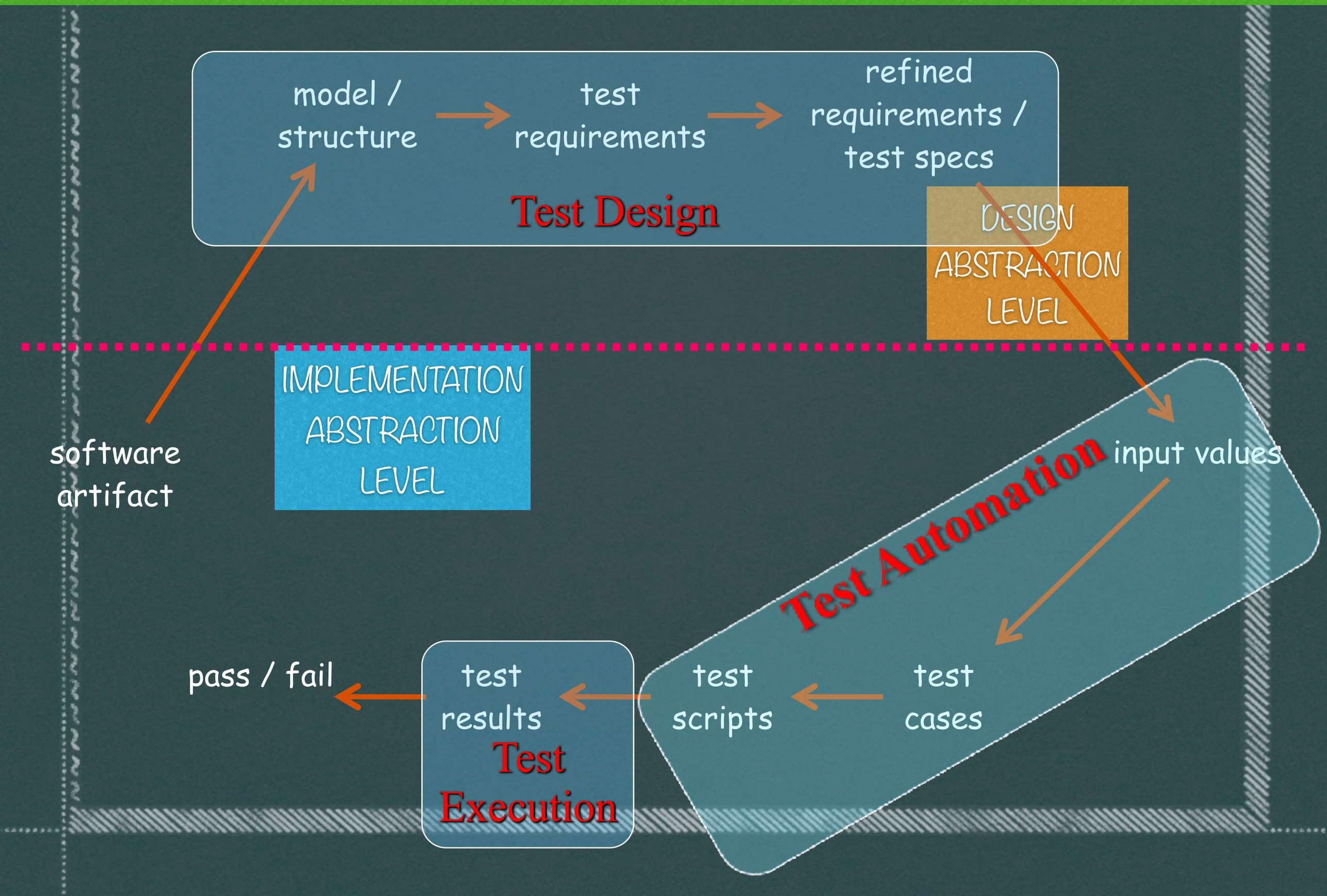
Model-Driven Test Design – Activities



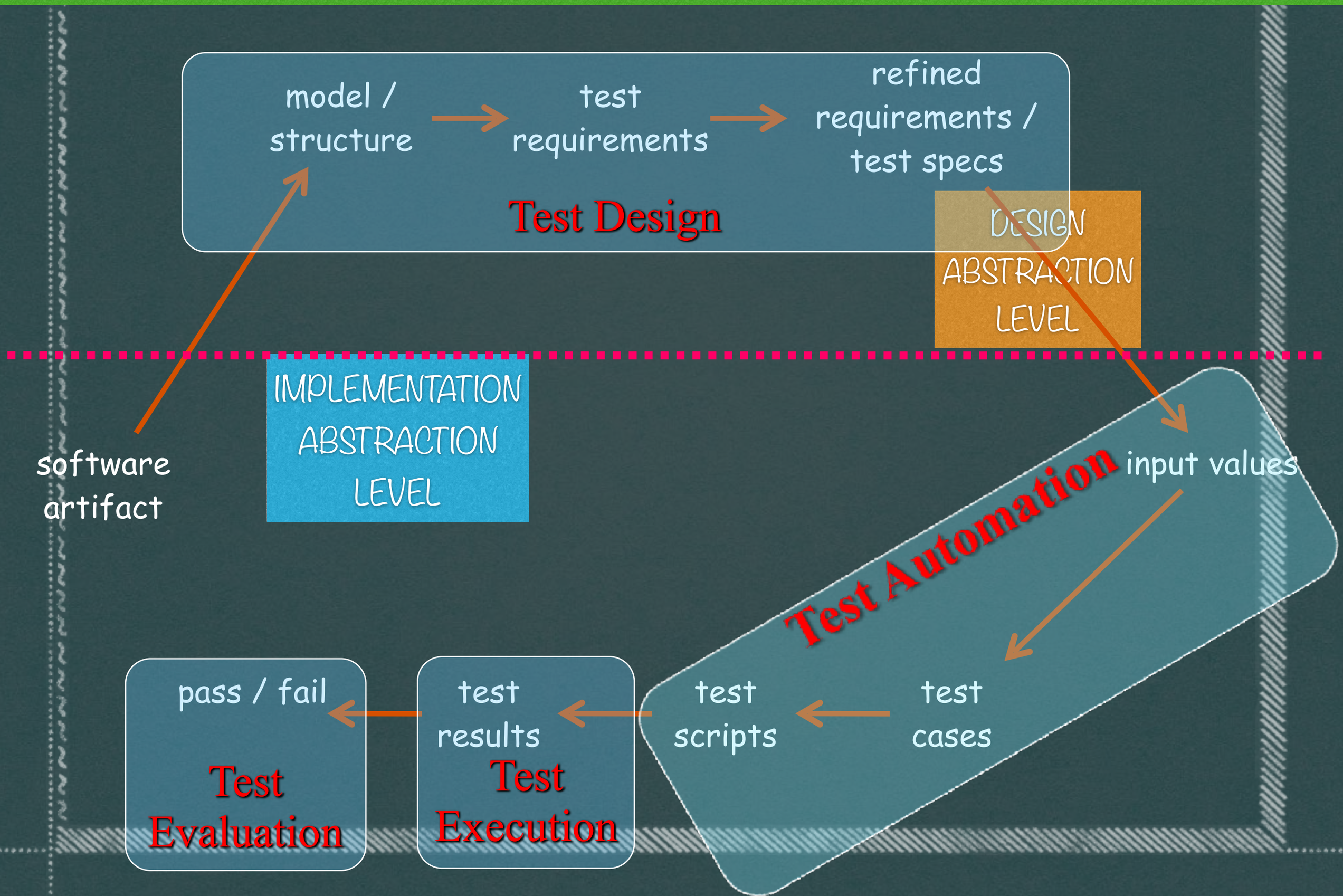
Model-Driven Test Design – Activities



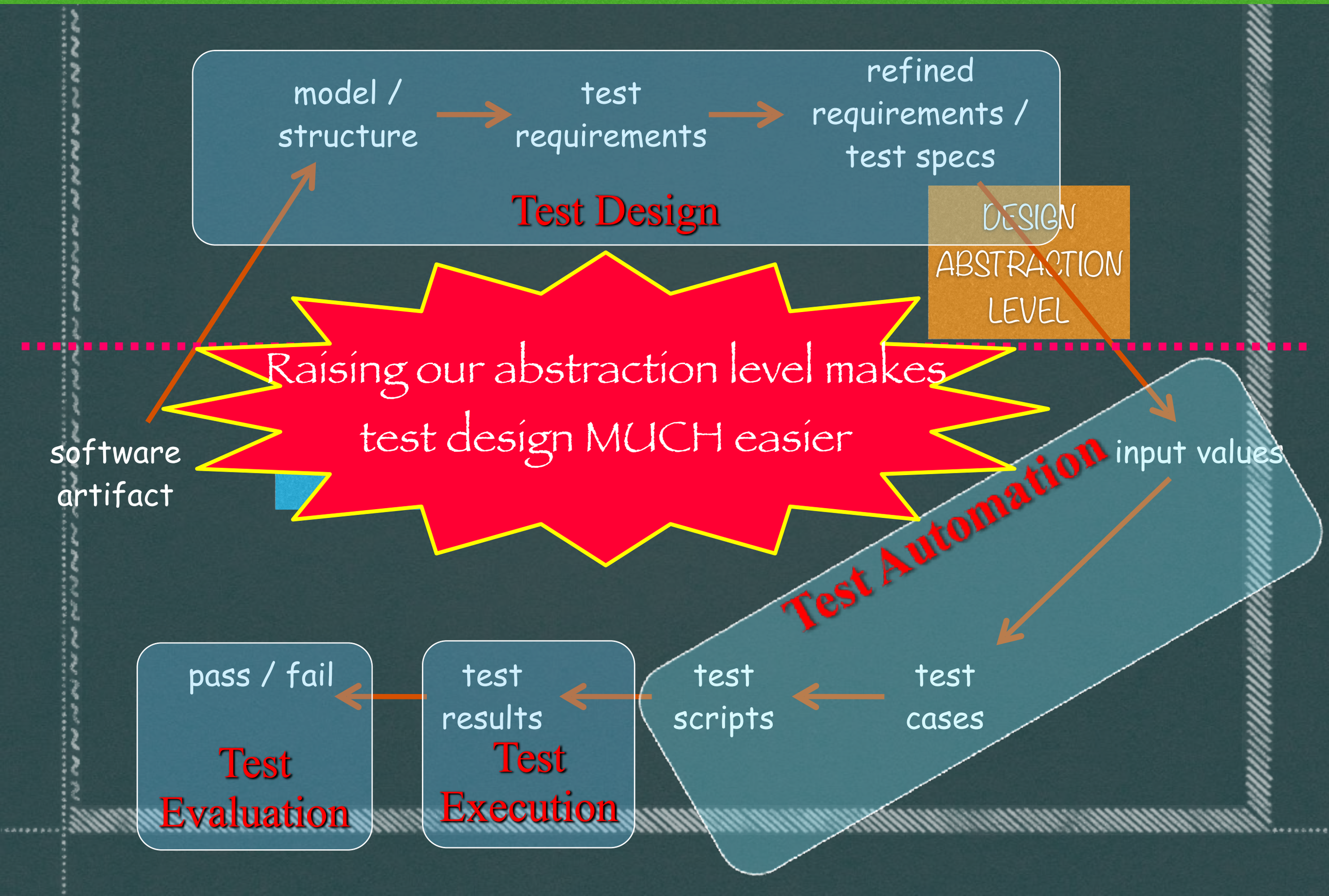
Model-Driven Test Design – Activities



Model-Driven Test Design – Activities



Model-Driven Test Design – Activities



واژگان آزمون نرم‌افزار

- مانند هر رشته‌ای، آزمون نرم‌افزار با تعداد زیادی واژگان همراه شده است.
- برخی از واژگان استاندارد هستند، برخی در مقالات و کتب و صنعت به صورت خاص مورد استفاده قرار گرفته‌اند و برخی تنها توسط افراد خاص مورد استفاده هستند.
- تعاریف کنونی در بر دارنده تعاریف معمول هستند.

واژگان مهم
Validation & Verification (*IEEE*)

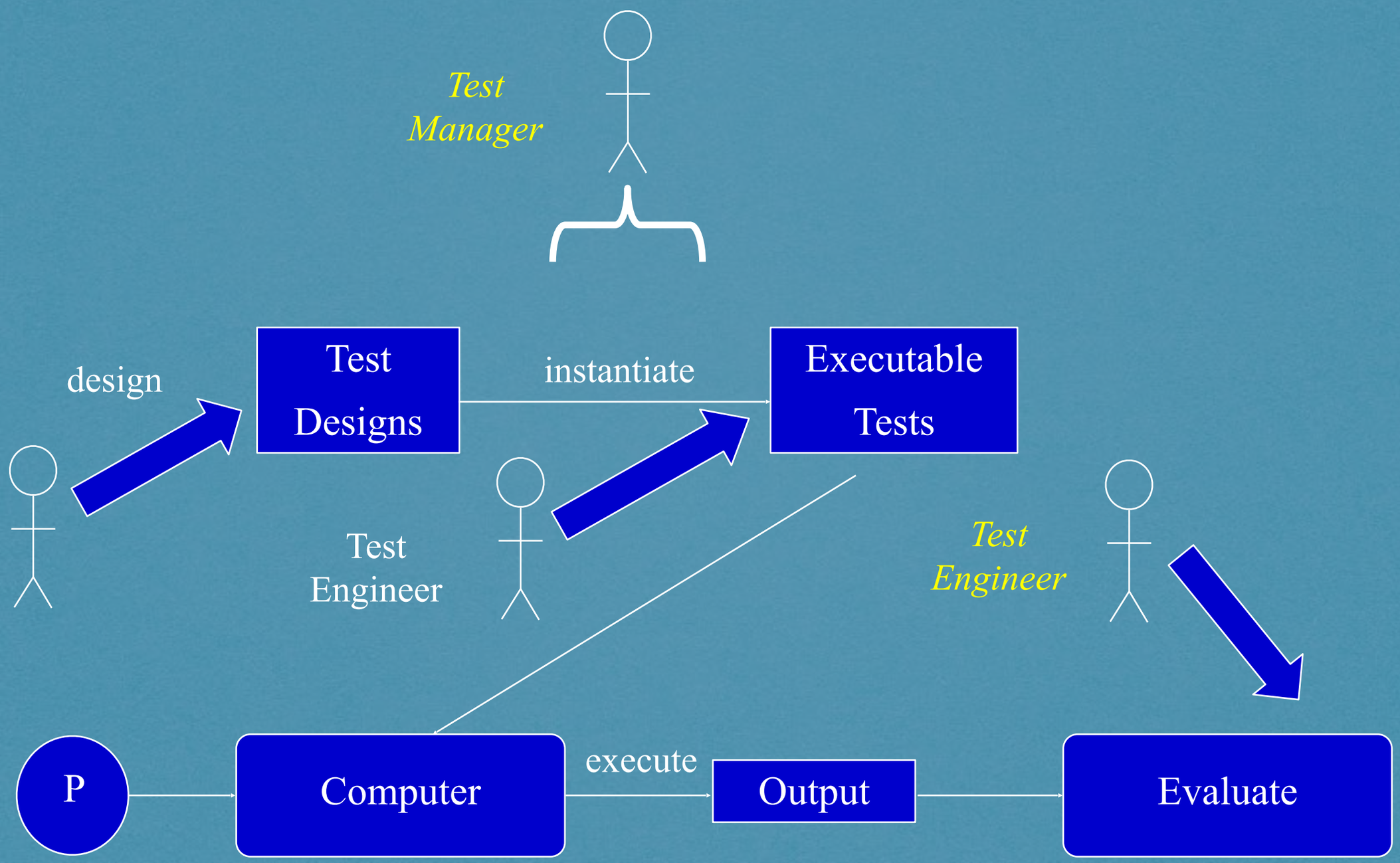
- Validation: فرآیند ارزیابی نرم‌افزار در پایان توسعه، برای اطمینان از تطابق با استفاده‌های تعریف شده برای آن
  - Are we building the right product
- Verification: فرآیند بررسی محصولات یک فاز و تطابق آن با نیازمندی‌های تعریف شده در فاز قبلی
  - Are we building the product right

IV&V stands for "*independent verification and validation*"

مهندس آزمون و مدیر آزمون

- Test Engineer : یک متخصص IT که مسئول امور تکنیکی‌ای مانند امور زیر است:
  - طراحی ورودی‌های آزمون
  - ایجاد مقادیر آزمون
  - اجرای اسکریپت‌های آزمون
  - تحلیل نتایج
  - گزارش نتایج به تیم توسعه
- Test Manager : مسئول یک یا چند مهندس آزمون:
  - وضع سیاست‌ها و رویه‌های آزمون
  - ارتباط با سایر مدیران پروژه
  - برخی اوقات: کمک به مهندسان آزمون در انجام کارهای زمین مانده!

فعالیت‌های مدیر آزمون



Static and Dynamic Testing

- Static Testing : آزمون نرم‌افزار بدون اجرای آن
  - شامل برخی از بررسی‌ها روی نرم‌افزار
  - برای یافتن مشکلات نرم‌افزار (و همچنین اشکالات بالقوه) بسیار موثر است
- Dynamic Testing : آزمون بر اساس مقادیر واقعی و تحت اجرا

Software Faults, Errors & Failures

- Software Fault : یک اشکال استاتیک در نرم‌افزار

- Software Error : داخلی، یک حالت غلط داخلی که بر اساس برخی اشکالات به وجود می‌آید.

- Software Failure : خارجی، یک نتیجه اشتباه از نرم‌افزار (بر اساس نیازمندی‌های تعریف شده برای آن)

Software Faults, Errors & Failures

- Software Fault : یک اشکال استاتیک در نرم‌افزار

Faults در نرم‌افزار مربوط به طراحی است و همیشه وجود دارد.

اساس برخی اشکالات به وجود می‌آید.

- Software Failure : خارجی، یک نتیجه اشتباه از نرم‌افزار (بر اساس نیازمندی‌های تعریف شده برای آن)

Testing & Debugging

- Testing : پیدا کردن ورودی‌هایی که سبب بروز خطا در نرم‌افزار می‌شود.

- Debugging : فرآیند پیدا کردن یک fault که سبب ایجاد failure در نرم‌افزار می‌شود.

Fault & Failure Model

سه شرط اساسی برای تبدیل fault به failure

1. Reachability : مکانی که خطا در آن وجود دارد باید قابل رسیدن باشد (قابل رسیدن باشد).
2. Infection : نرم‌افزار باید به حالت غلطی رود (عفونی شود).
3. Propagation : حالت غلط باید منتشر شود و این انتشار سبب ایجاد خروجی‌های اشتباه شود. (به خروجی منتشر شود)

Test Case

- Test Case Values : مقداری که یک نیازمندی آزمون را برآورده می کند.

- Expected Results : مقداری که پس از اجرای یک آزمون باید در خروجی دیده شود. (با توجه به وظیفه و رفتار تعریف شده برای نرم‌افزار)

Observability and Controllability

- Software Observability : رفتار نرم‌افزار و تاثیرات آن روی محیط و سخت‌افزار چقدر قابل مشاهده است؟
  - نرم‌افزارهایی که روی سخت‌افزار، پایگاه داده و remote file تاثیر می‌گذارند دارای قابلیت مشاهده پایینی هستند.
- Software Controllability : چقدر می‌توان یک برنامه را با توجه به ورودی‌های آن کنترل کرد؟ کنترل شامل مقادیر، عملگرها و رفتارها هستند.
  - راحتی کنترل از طریق ورودی‌های کیبورد
  - ورودی از سنسورهای سخت‌افزاری و یا نرم‌افزارها سخت‌تر است
  - انتزاع داده‌ای، کنترل‌پذیری و قابلیت مشاهده را پایین می‌آورد.

مقادیر تاثیر گذار روی Controllability و Observability

- Prefix Values : برخی مقادیر برای قرار دادن نرم‌افزار در حالتی که بتواند مقادیر مورد آزمون را دریافت کند.
- Postfix Values : مقادیری که لازم است قرار داده شوند تا نتیجه آزمون در خروجی مشاهده شود.

- **دو نوع مقدار postfix:**

1. Verification Values : مقادیری که برای مشاهده نتایج مورد آزمون در خروجی مورد نیاز است.
2. Exit Commands : مقادیری که برای خروج از نرم‌افزار و یا قرار گرفتن در حالت پایا لازم است.

- Executable Test Script : یک مورد آزمون که قابل اجرا است و می‌تواند از مقادیر خروجی نرم‌افزار گزارش ثبت کند.

Top-Down and Bottom-Up Testing

- Top-Down Testing : ابتدا آزمون دامنه و سپس آزمون‌های سطح پایین‌تر با توجه به توالی صدا زدن رویه‌ها
- Bottom-Up Testing : ابتدا برگ‌ها (رویه‌هایی که هیچ رویه دیگری را صدا نمی‌زنند) و بالا رفتن به سمت ریشه
- تا زمانی که همه بچه‌های یک رویه مورد آزمون قرار نگیرند، آن رویه مورد آزمون قرار نمی‌گیرد.

White-box and Black-box Testing

- Black-box testing : ایجاد آزمون از روی توصیف خارجی نرم‌افزار، نیازمندی‌ها و طراحی آن
- White-box testing : ایجاد آزمون از روی کد و حالات داخلی نرم‌افزار مانند شاخه‌ها، شرایط خاص و جملات

White-box and Black-box Testing

- Black-box testing: ایجاد آزمون از روی توصیف خارجی نرم‌افزار، نیازمندی‌ها و طراحی آن

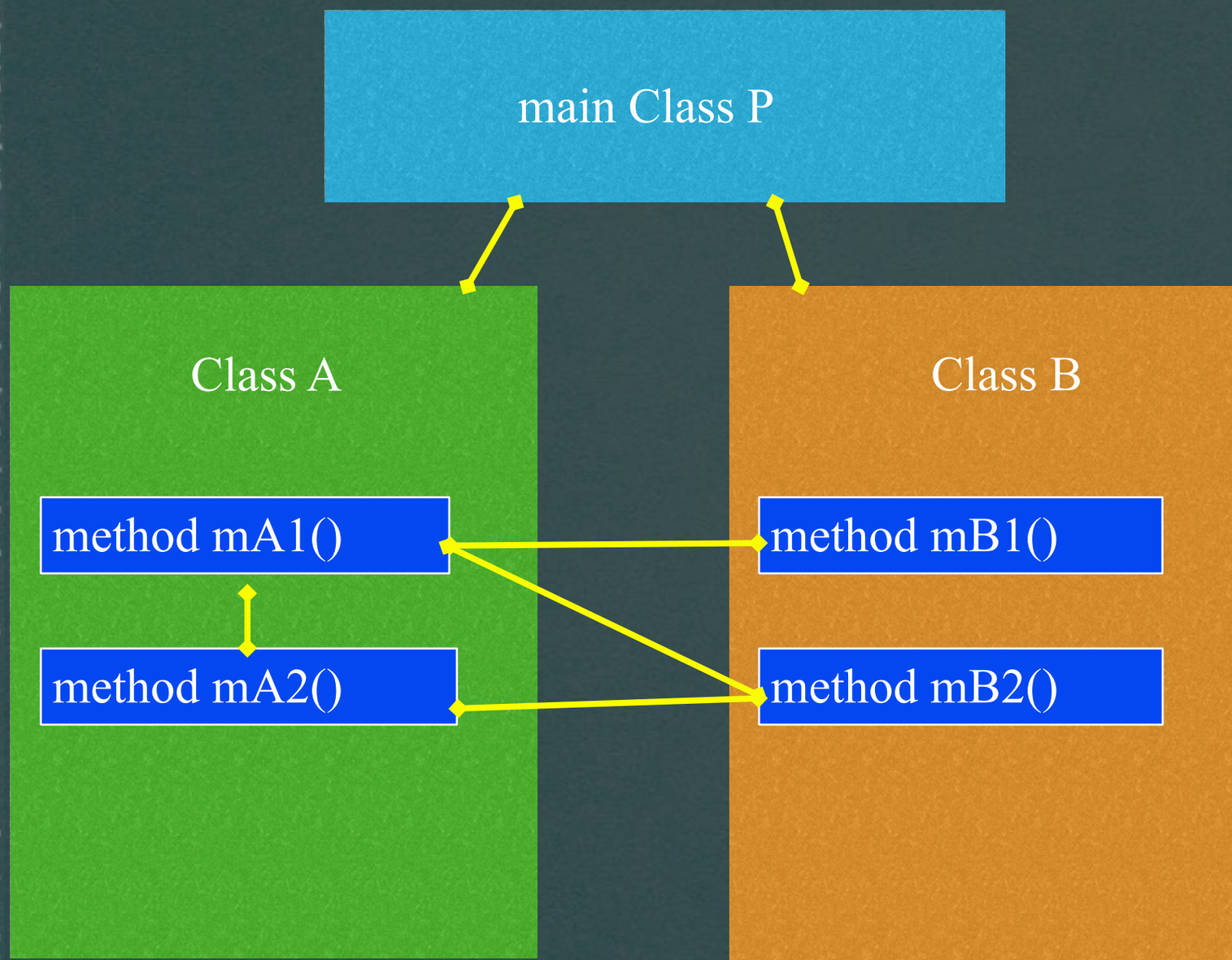
این دیدگاه منسوخ شده است! یک سوال کلی‌تر این است که: از چه سطحی از انتزاع می‌خوایم آزمون‌ها را ایجاد کنیم؟

- White-box testing: ایجاد آزمون از روی کد و حالات داخلی نرم‌افزار مانند شاخه‌ها، شرایط خاص و جملات

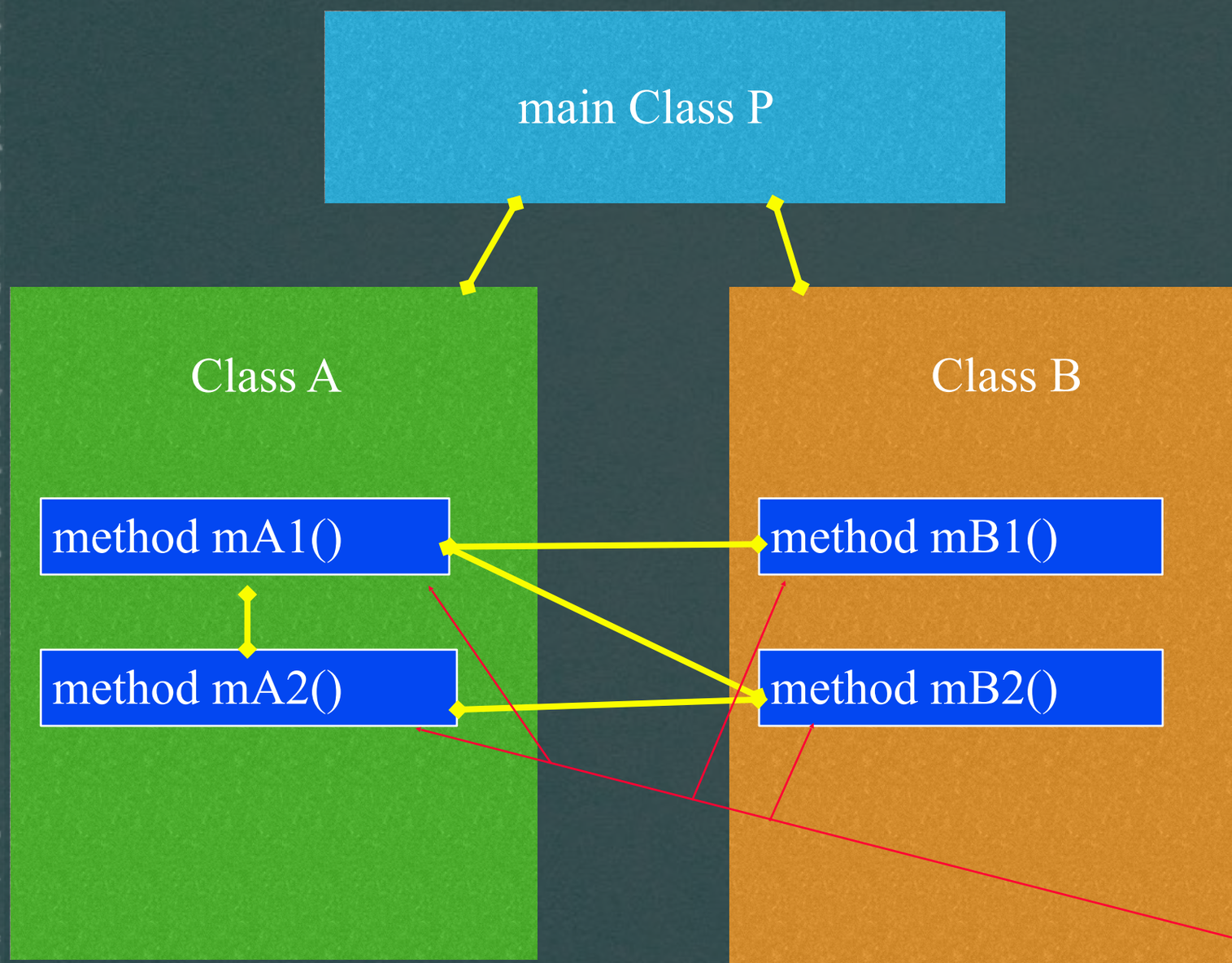
تغییر مفاهیم آزمون

- نگاه قدیمی به آزمون: آزمون در یک فاز خاص توسعه بوده است.
  - Unit, module, integration, system ...
- نگاه جدید به آزمون: آزمون بر اساس ساختار و معیار
  - گراف، عبارات منطقی، نحو و فضای دامنه

Old : Testing at Different Levels

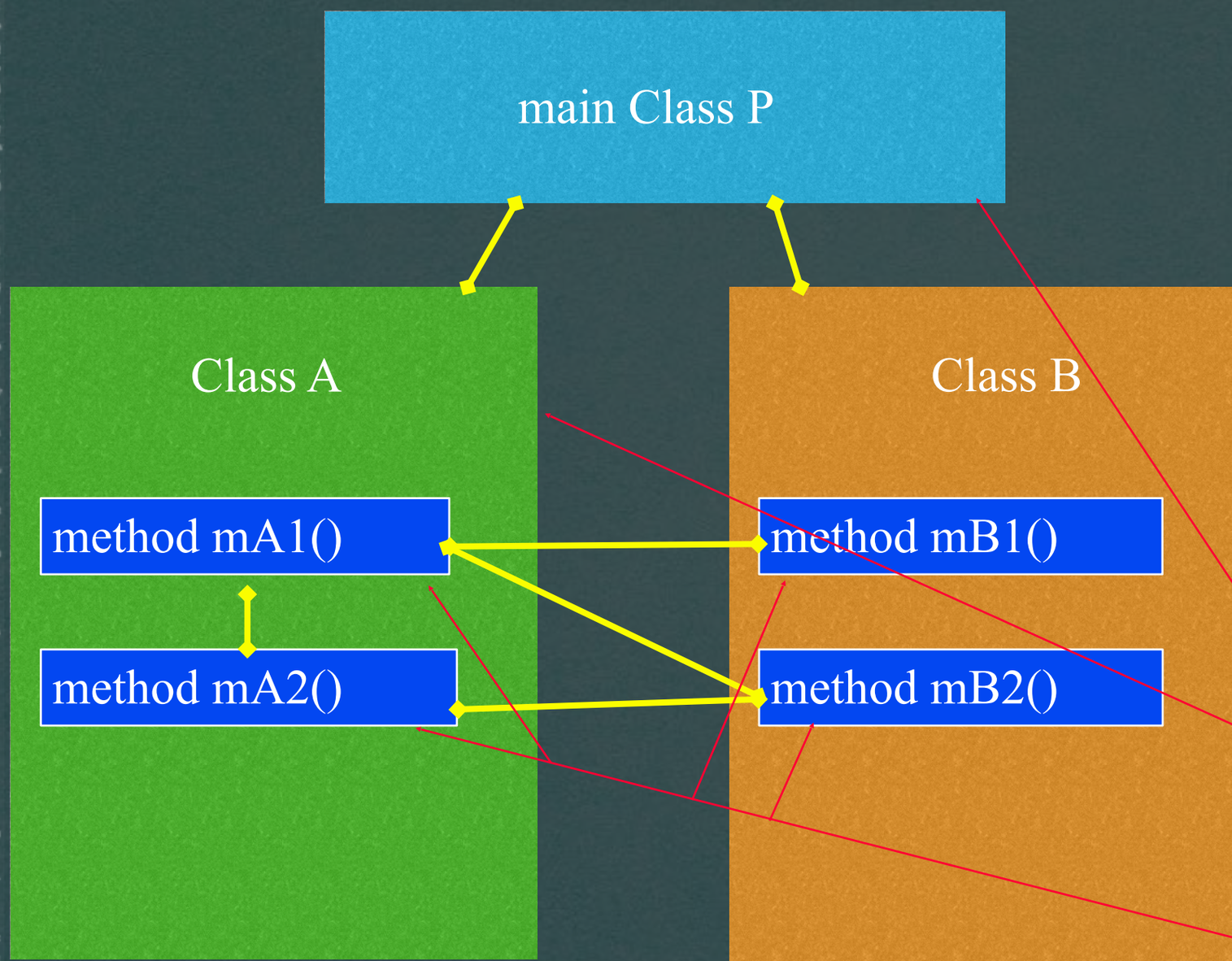


Old : Testing at Different Levels



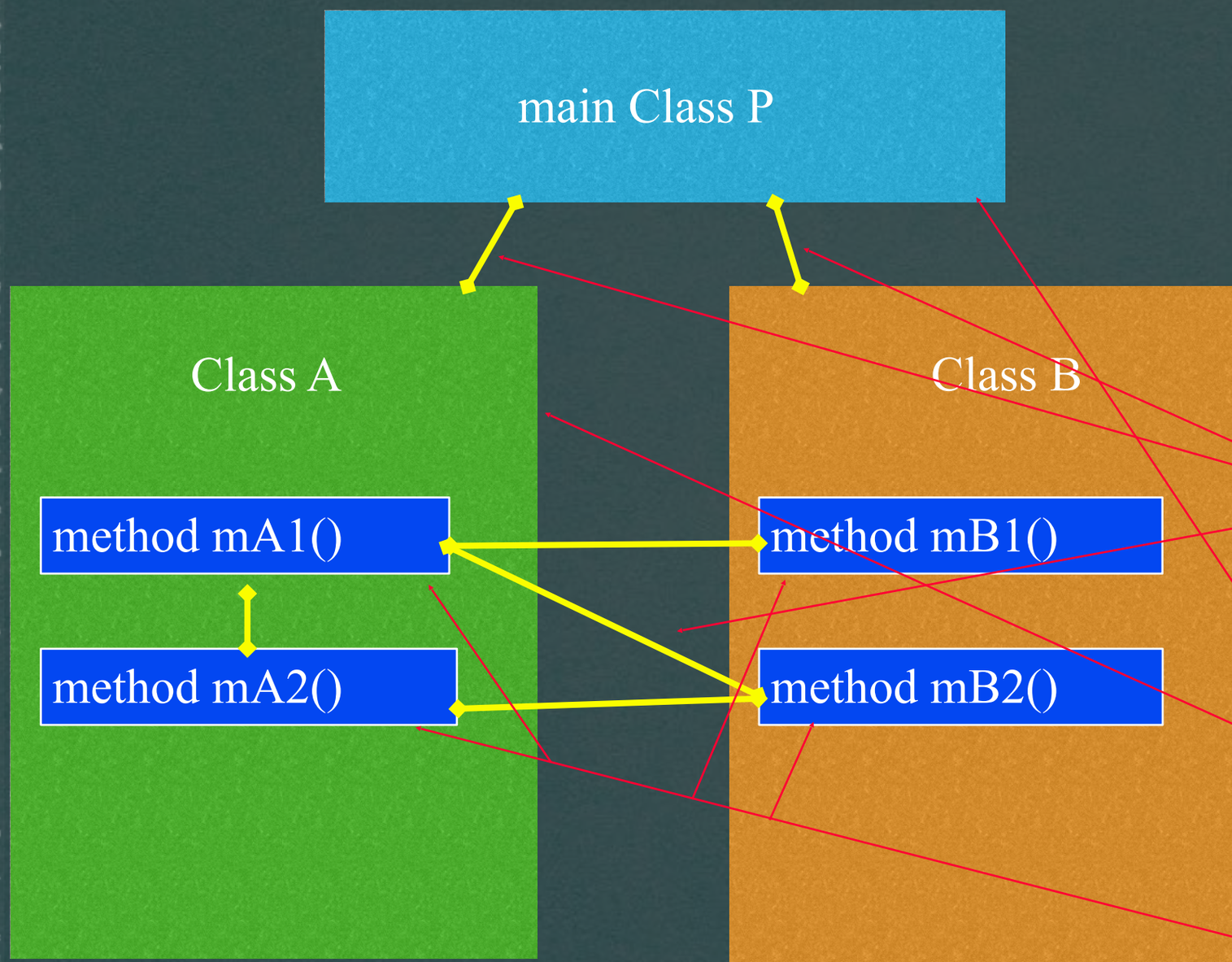
- Unit testing: Test each unit (method) individually

Old : Testing at Different Levels



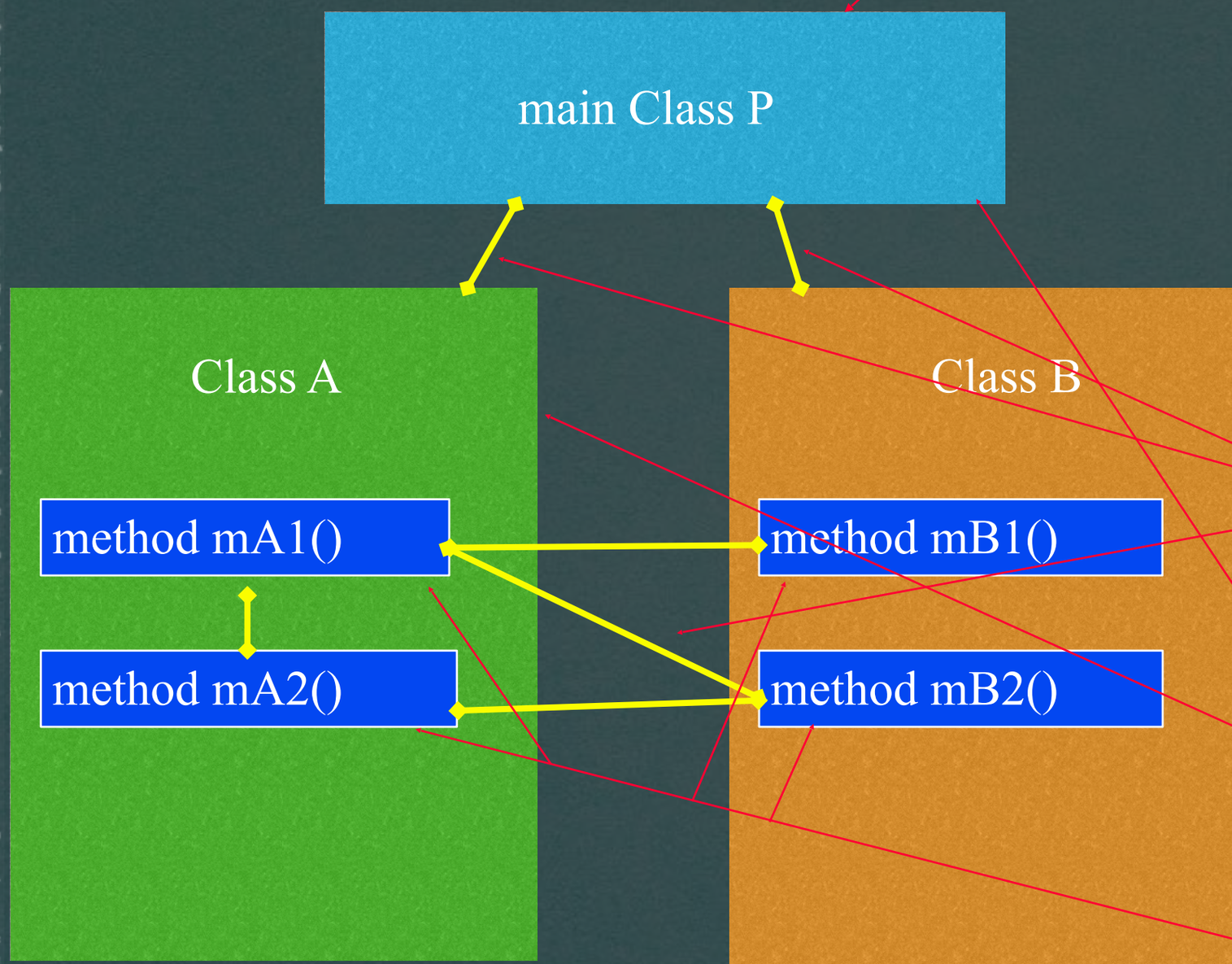
- Module testing: Test each class, file, module or component
- Unit testing: Test each unit (method) individually

Old : Testing at Different Levels



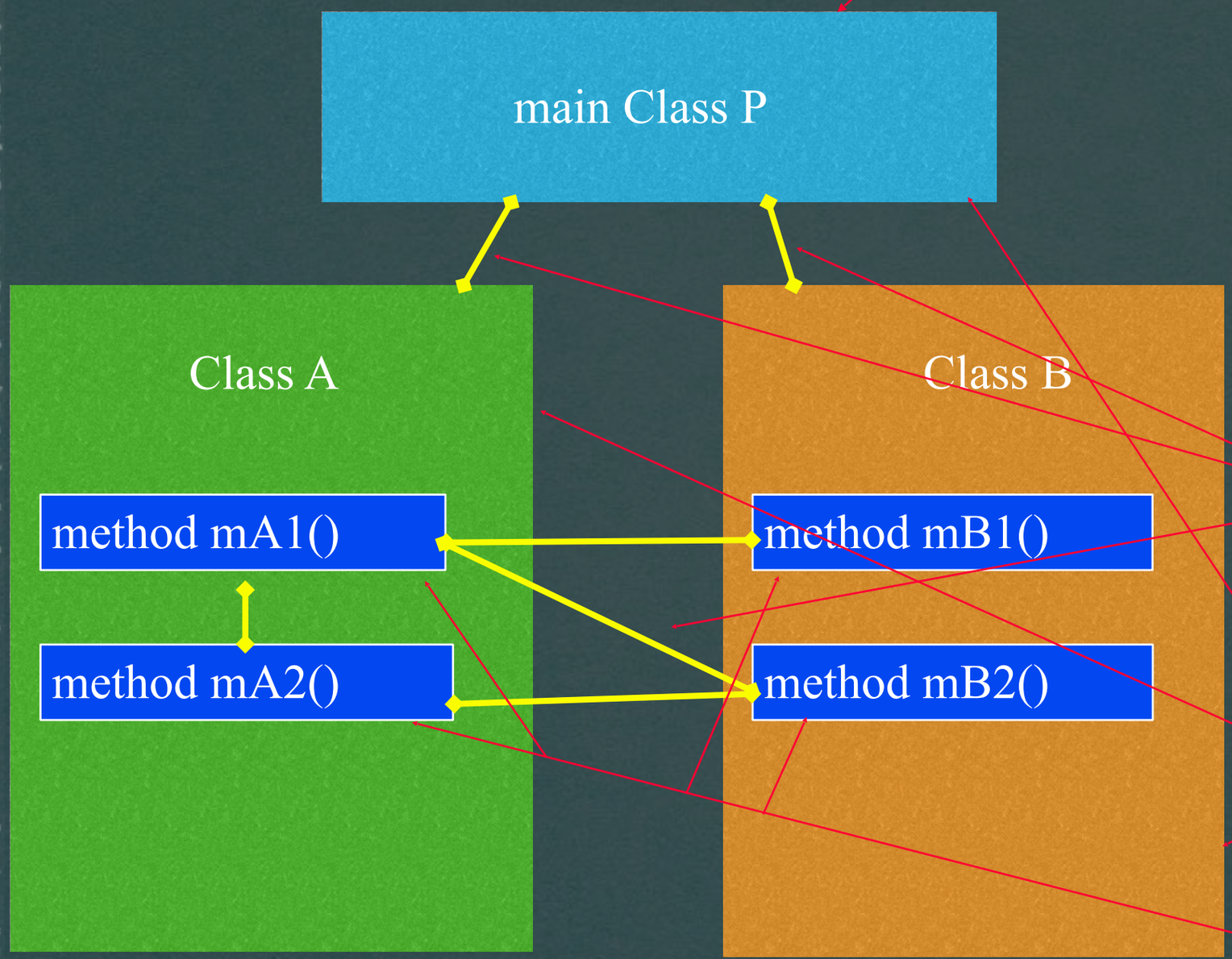
- Integration testing: Test how modules interact with each other
- Module testing: Test each class, file, module or component
- Unit testing: Test each unit (method) individually

Old : Testing at Different Levels



- System testing: Test the overall functionality of the system
- Integration testing: Test how modules interact with each other
- Module testing: Test each class, file, module or component
- Unit testing: Test each unit (method) individually

Old : Testing at Different Levels



- Acceptance testing: Is the software acceptable to the user?
- System testing: Test the overall functionality of the system
- Integration testing: Test how modules interact with each other
- Module testing: Test each class, file, module or component
- Unit testing: Test each unit (method) individually

Old : Find a Graph and Cover It

- مناسب:
  - یک مصنوع خاص نرم‌افزار
    - کد، طراحی، نیازمندی‌ها
  - یک فاز خاص از توسعه
    - نیامندی‌ها، توصیف، طراحی، پیاده‌سازی

- گراف به تنهایی تمامی روش‌های آزمون را پوشش نمی‌دهد.
  - کفایت چهار سطح انتزاع مدل‌ها

New : Test Coverage Criteria

وظیفه آزمونگر ساده است:

تعریف یک مدل از نرم‌افزار و ایجاد آزمون‌هایی برای پوشاندن آن

- **Test Requirements : مشخص کردن مواردی که باید در آزمون پوشانده شوند.**
- **Test Criterion : مجموعه‌ای از اصول و قوانین و یک رویه که نیازمندی‌های آزمون را تعریف می‌کند.**

New : Test Coverage Criteria

وظیفه آزمونگر ساده است:

تعریف یک مدل از نرم‌افزار و ایجاد آزمون‌هایی برای پوشاندن آن

- **Test Requirements : مشخص کردن مواردی که باید در آزمون پوشانده شوند.**
- **Test Criterion : مجموعه‌ای از اصول و قوانین و یک رویه که نیازمندی‌های آزمون را تعریف می‌کند.**

محققین معیارهای زیادی برای آزمون تعریف کرده‌اند، ولی در عمل تمامی آن‌ها معیارهایی روی چهار ساختار کلی هستند.

New : Criteria Based on Structures

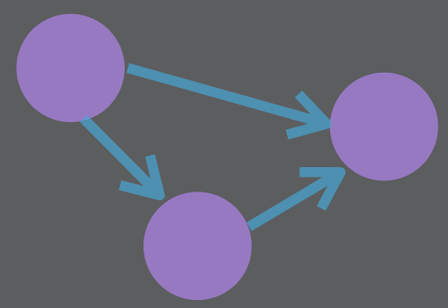
ساختار: چهار روش برای مدل کردن نرم‌افزار

1. Graphs
2. Logical Expressions
3. Input Domain Characterization
4. Syntactic Structures

New : Criteria Based on Structures

ساختار: چهار روش برای مدل کردن نرم‌افزار

1. Graphs



2. Logical Expressions

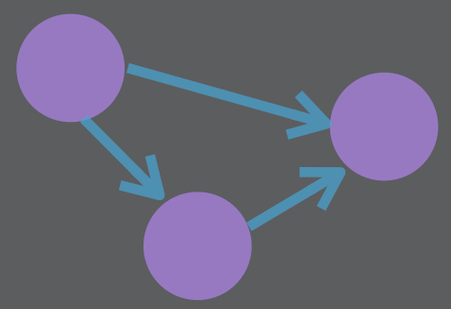
3. Input Domain Characterization

4. Syntactic Structures

New : Criteria Based on Structures

ساختار: چهار روش برای مدل کردن نرم‌افزار

1. Graphs



2. Logical Expressions

(not X or not Y) and A and B

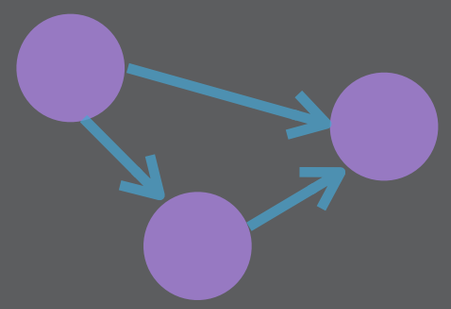
3. Input Domain Characterization

4. Syntactic Structures

New : Criteria Based on Structures

ساختار: چهار روش برای مدل کردن نرم‌افزار

1. Graphs



2. Logical Expressions

(not X or not Y) and A and B

3. Input Domain Characterization

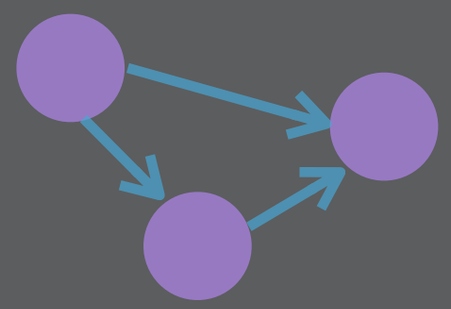
A: {0, 1, >1}
B: {600, 700, 800}
C: {swe, cs, isa, infs}

4. Syntactic Structures

New : Criteria Based on Structures

ساختار: چهار روش برای مدل کردن نرم‌افزار

1. Graphs



2. Logical Expressions

(not X or not Y) and A and B

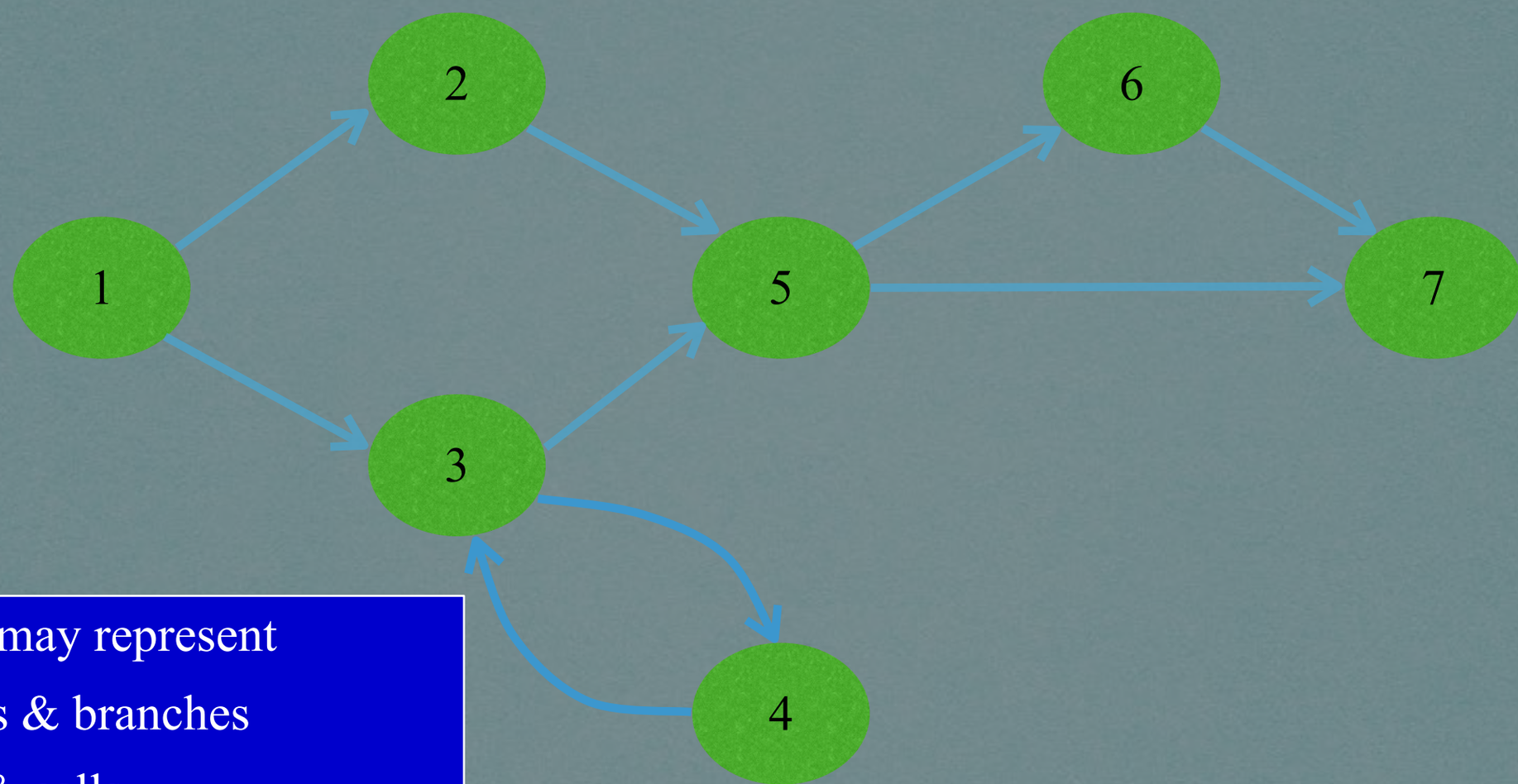
3. Input Domain Characterization

A: {0, 1, >1}
B: {600, 700, 800}
C: {swe, cs, isa, infs}

4. Syntactic Structures

```
if (x > y)
    z = x - y;
else
   z = 2 * x;
```

1. Graph Coverage – Structural

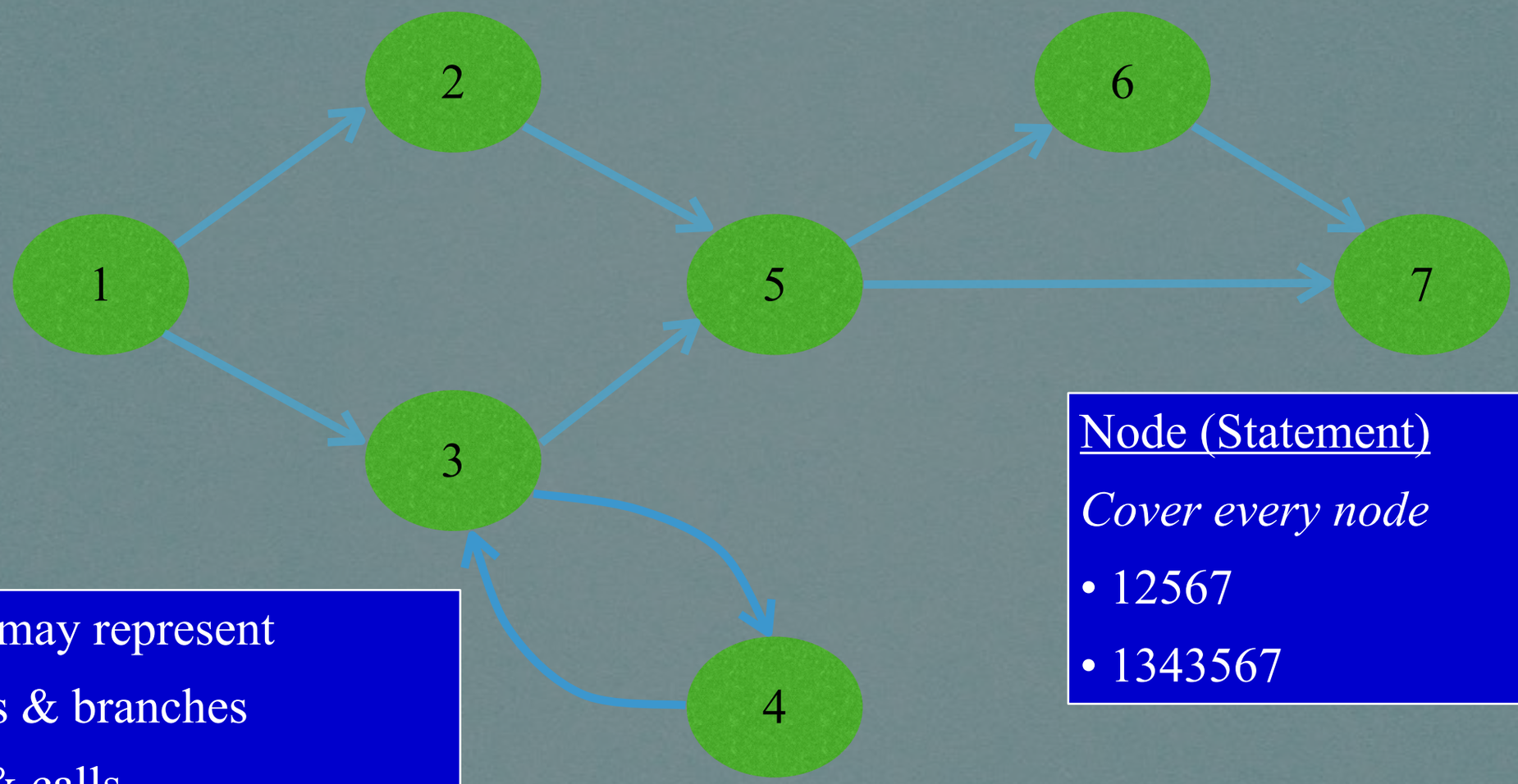


This graph may represent

- statements & branches
- methods & calls
- components & signals
- states and transitions

⋮

1. Graph Coverage – Structural



This graph may represent

- statements & branches
- methods & calls
- components & signals
- states and transitions

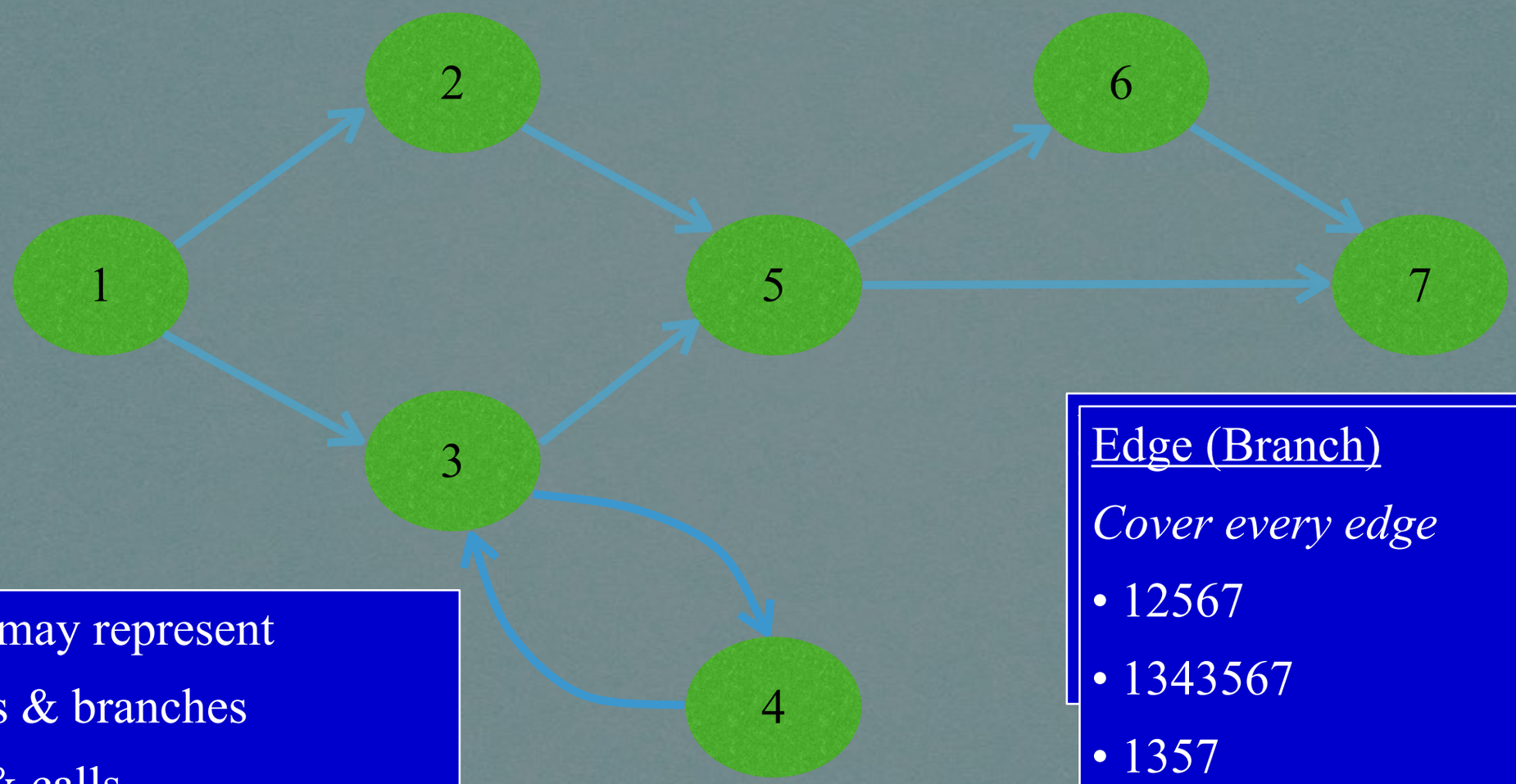
⋮

Node (Statement)

Cover every node

- 12567
- 1343567

1. Graph Coverage – Structural



This graph may represent

- statements & branches
- methods & calls
- components & signals
- states and transitions

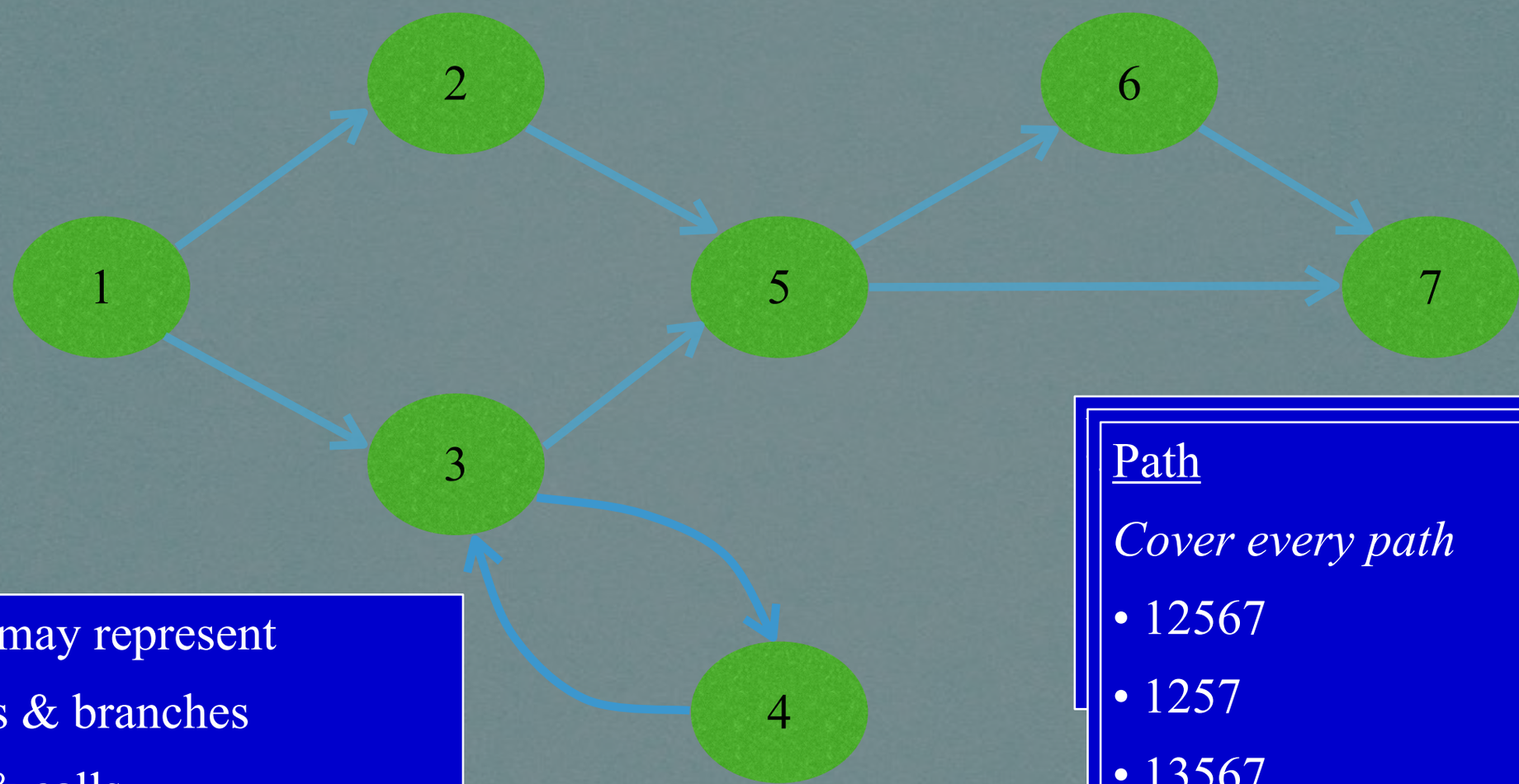
⋮

<u>Edge (Branch)</u>

Cover every edge

- 12567
- 1343567
- 1357

1. Graph Coverage – Structural



This graph may represent

- statements & branches
- methods & calls
- components & signals
- states and transitions

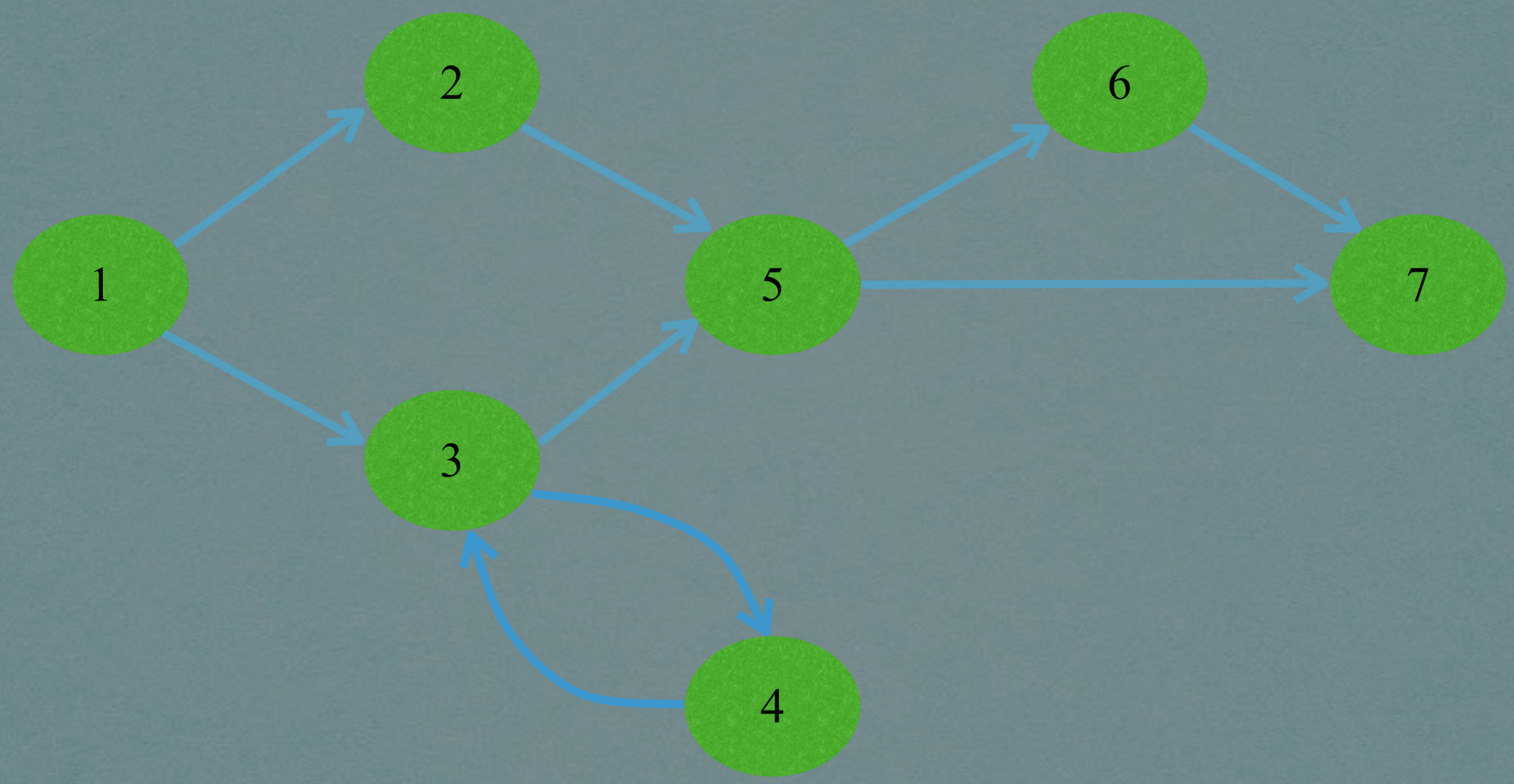
⋮

Path

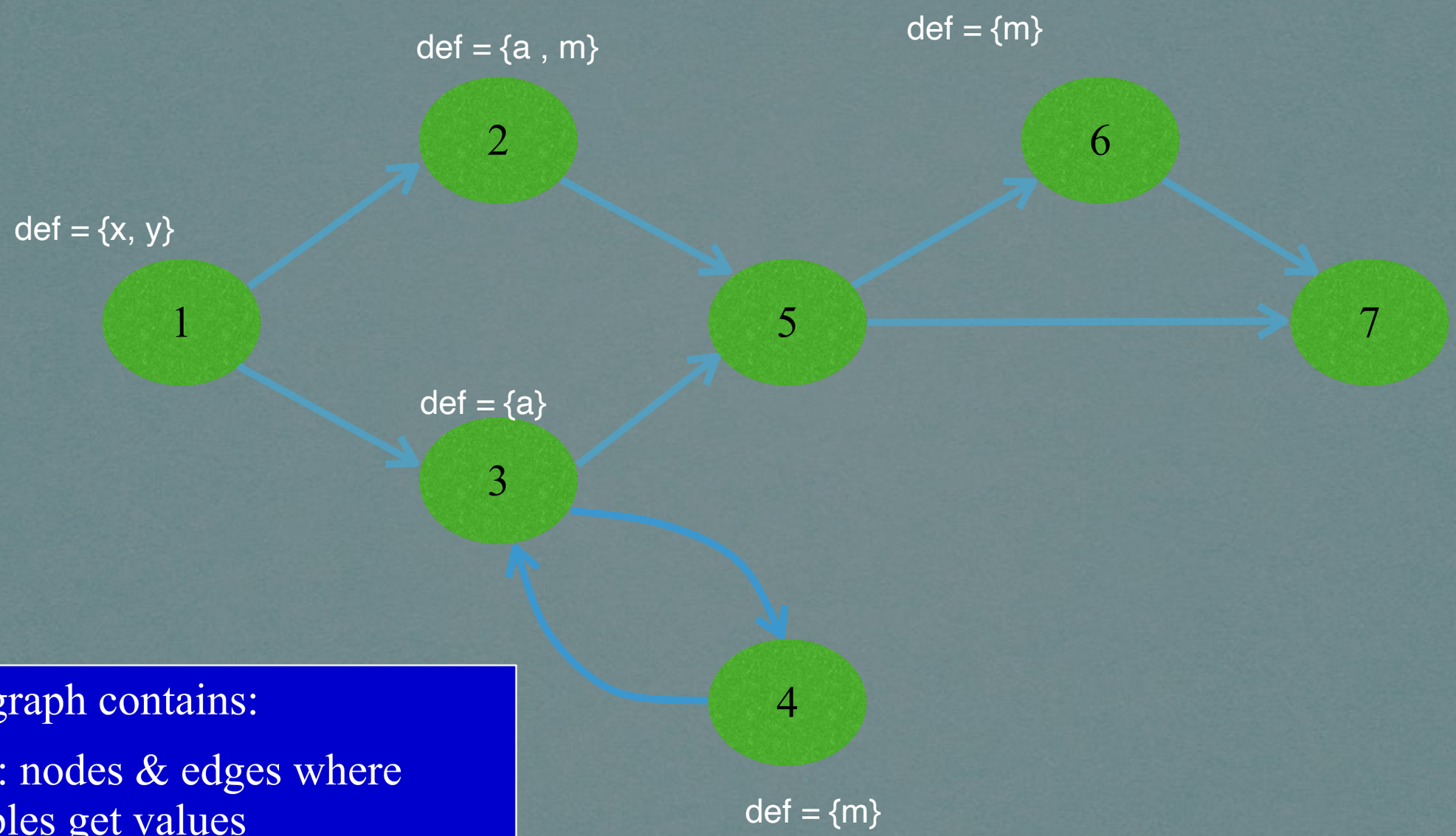
Cover every path

- 12567
- 1257
- 13567
- 1357
- 1343567
- 134357 …

1. Graph Coverage – Data Flow



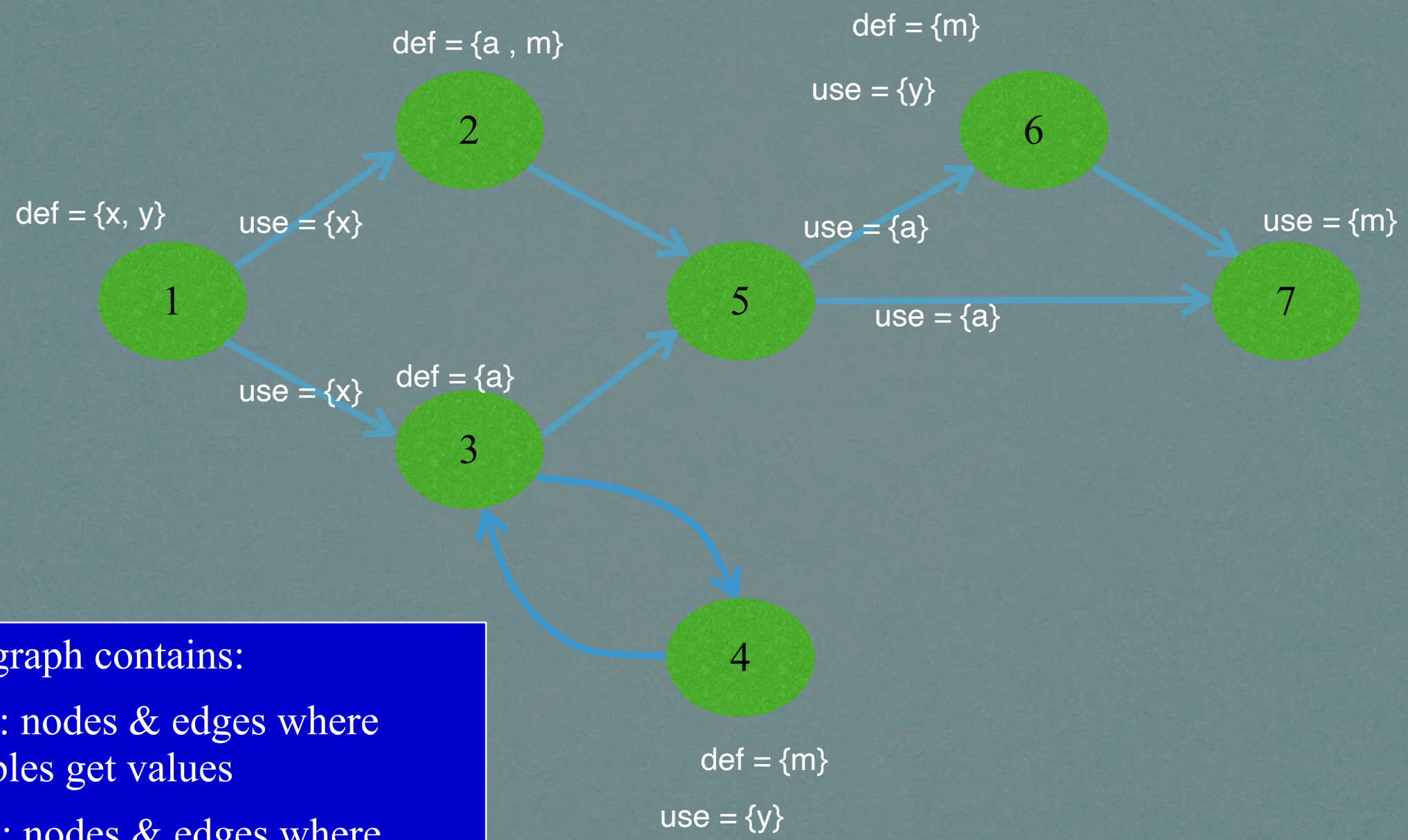
1. Graph Coverage – Data Flow



This graph contains:

- <u>defs</u>: nodes & edges where variables get values
- <u>uses</u>: nodes & edges where values are accessed

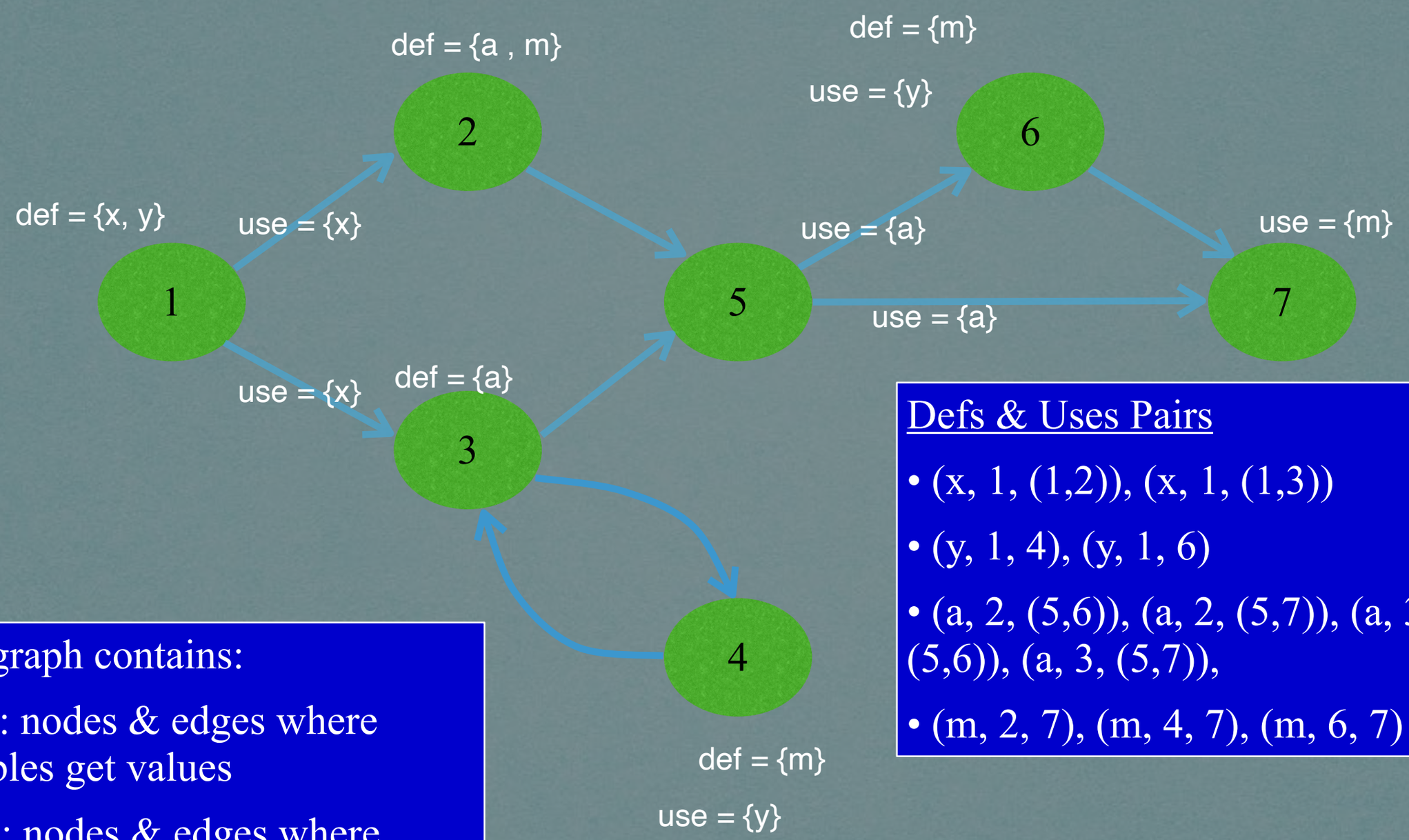
1. Graph Coverage – Data Flow



This graph contains:

- <u>defs</u>: nodes & edges where variables get values
- <u>uses</u>: nodes & edges where values are accessed

1. Graph Coverage – Data Flow



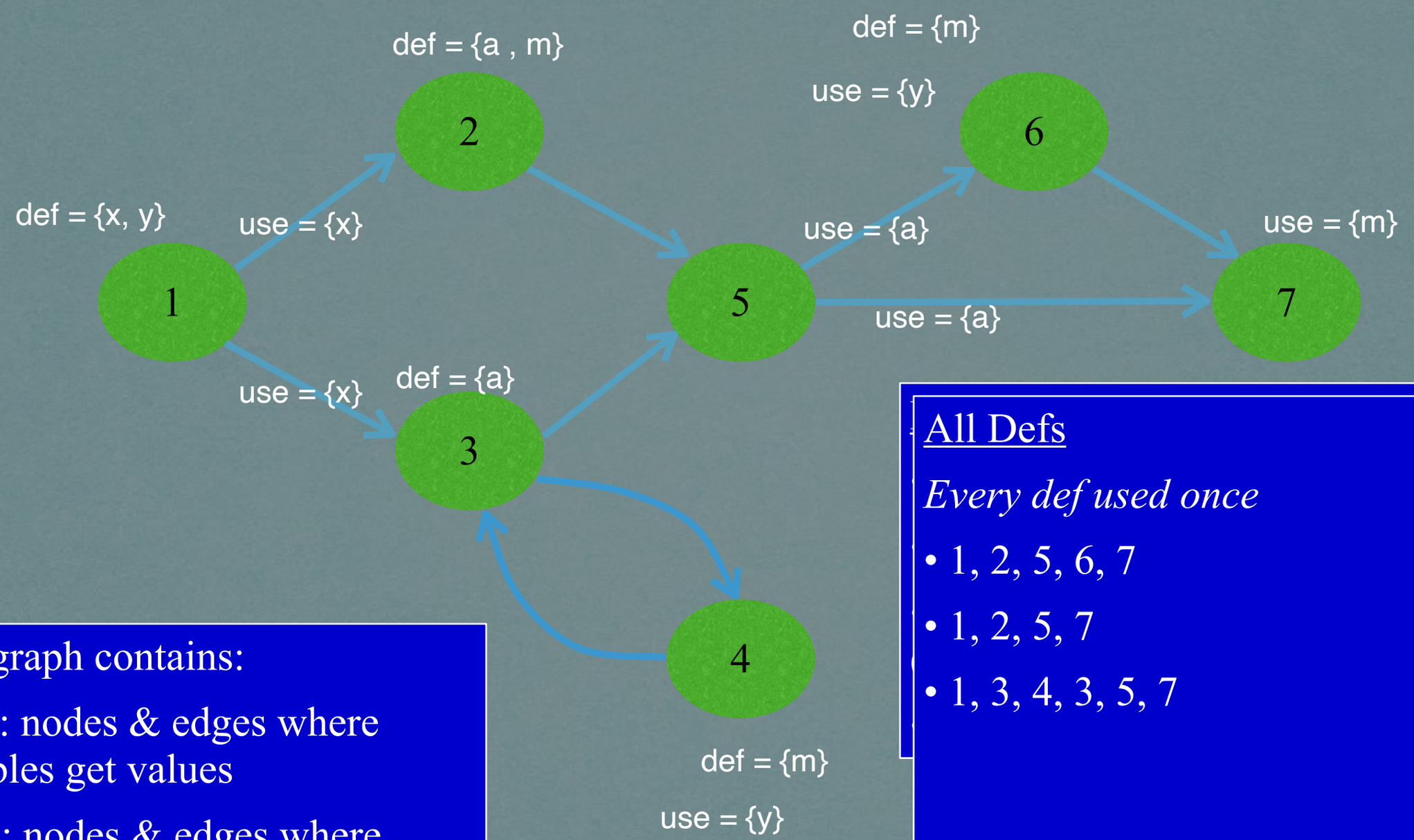
This graph contains:

- defs: nodes & edges where variables get values
- uses: nodes & edges where values are accessed

Defs & Uses Pairs

- (x, 1, (1,2)), (x, 1, (1,3))
- (y, 1, 4), (y, 1, 6)
- (a, 2, (5,6)), (a, 2, (5,7)), (a, 3, (5,6)), (a, 3, (5,7)),
- (m, 2, 7), (m, 4, 7), (m, 6, 7)

1. Graph Coverage – Data Flow



This graph contains:

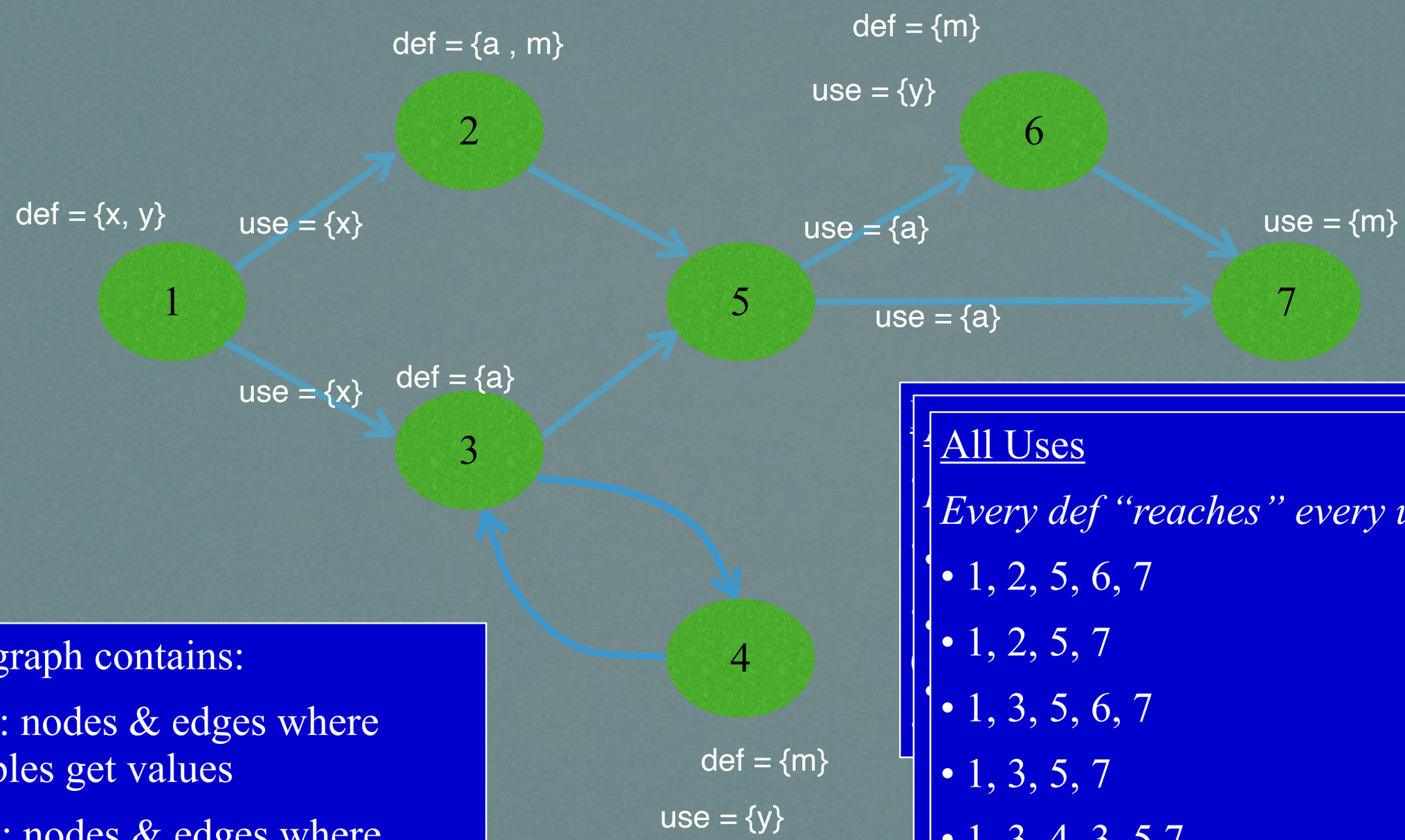
- <u>defs</u>: nodes & edges where variables get values
- <u>uses</u>: nodes & edges where values are accessed

<u>All Defs</u>

Every def used once

- 1, 2, 5, 6, 7
- 1, 2, 5, 7
- 1, 3, 4, 3, 5, 7

1. Graph Coverage – Data Flow



This graph contains:

- <u>defs</u>: nodes & edges where variables get values
- <u>uses</u>: nodes & edges where values are accessed

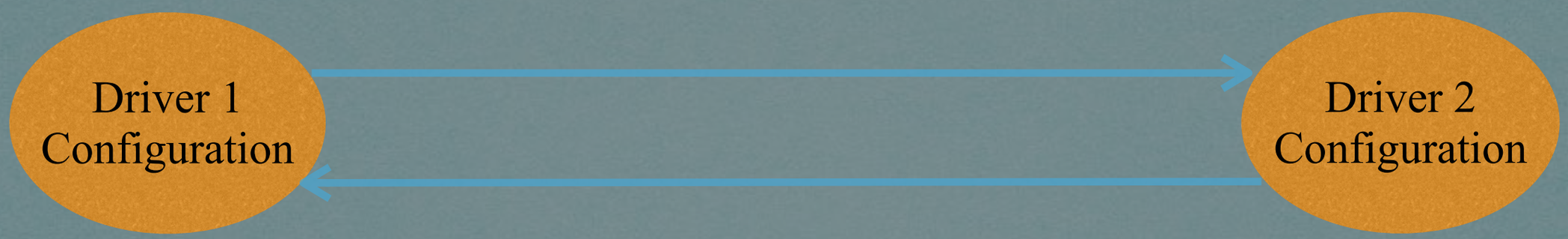
<u>All Uses</u>

Every def "reaches" every use

- 1, 2, 5, 6, 7
- 1, 2, 5, 7
- 1, 3, 5, 6, 7
- 1, 3, 5, 7
- 1, 3, 4, 3, 5,7

1. Graph – FSM Example

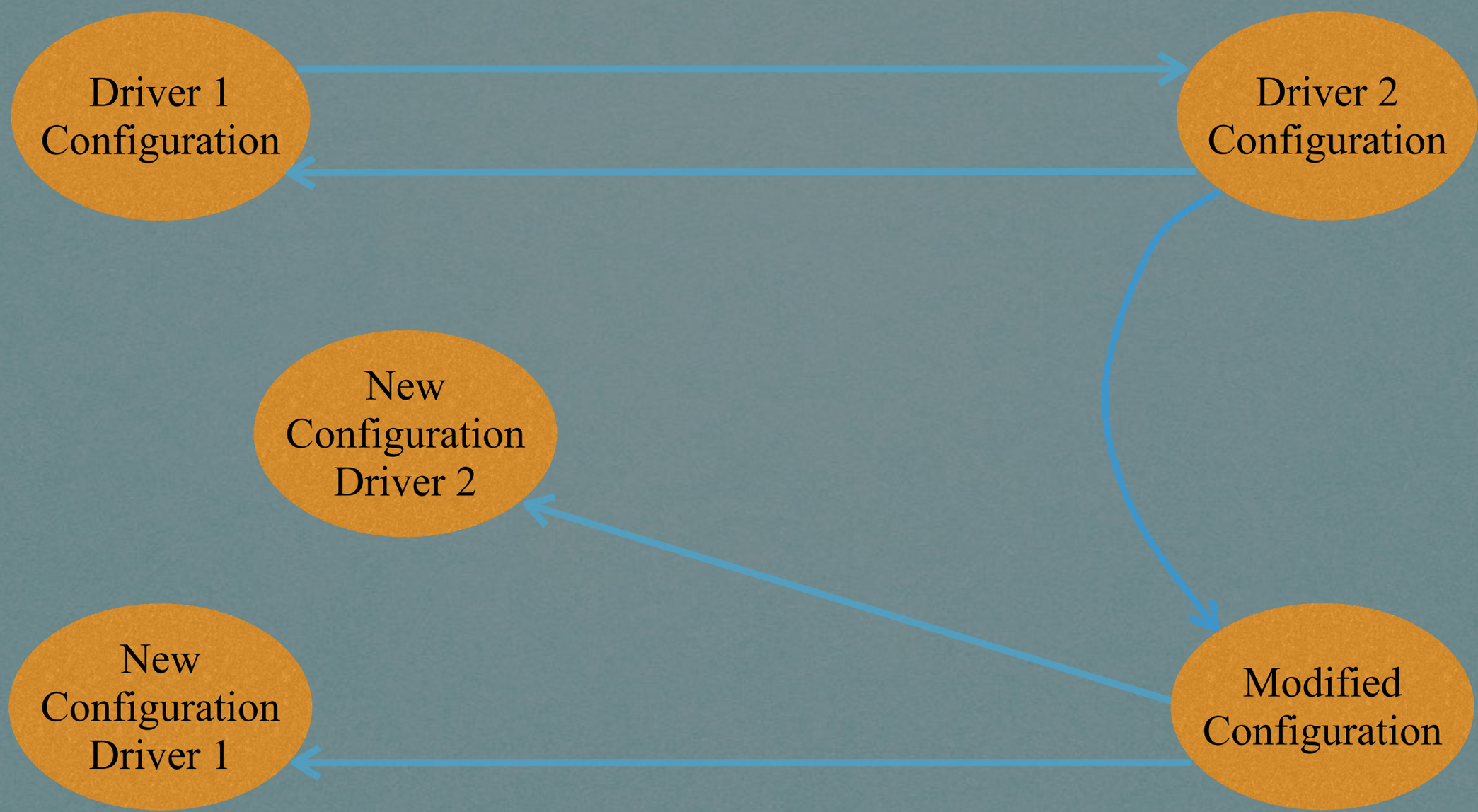
Memory Seats in a Lexus ES 350



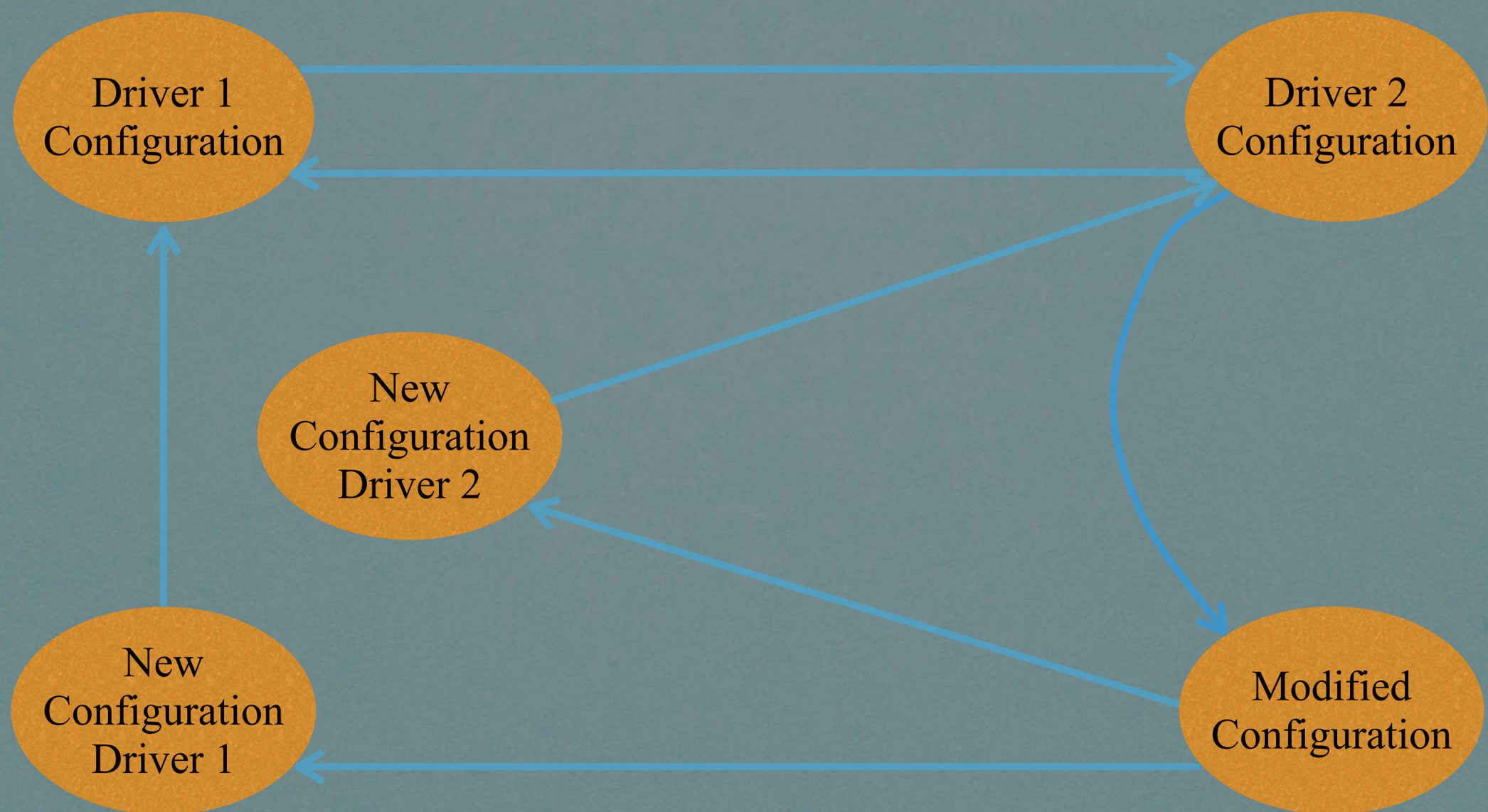
1. Graph – FSM Example

Memory Seats in a Lexus ES 350

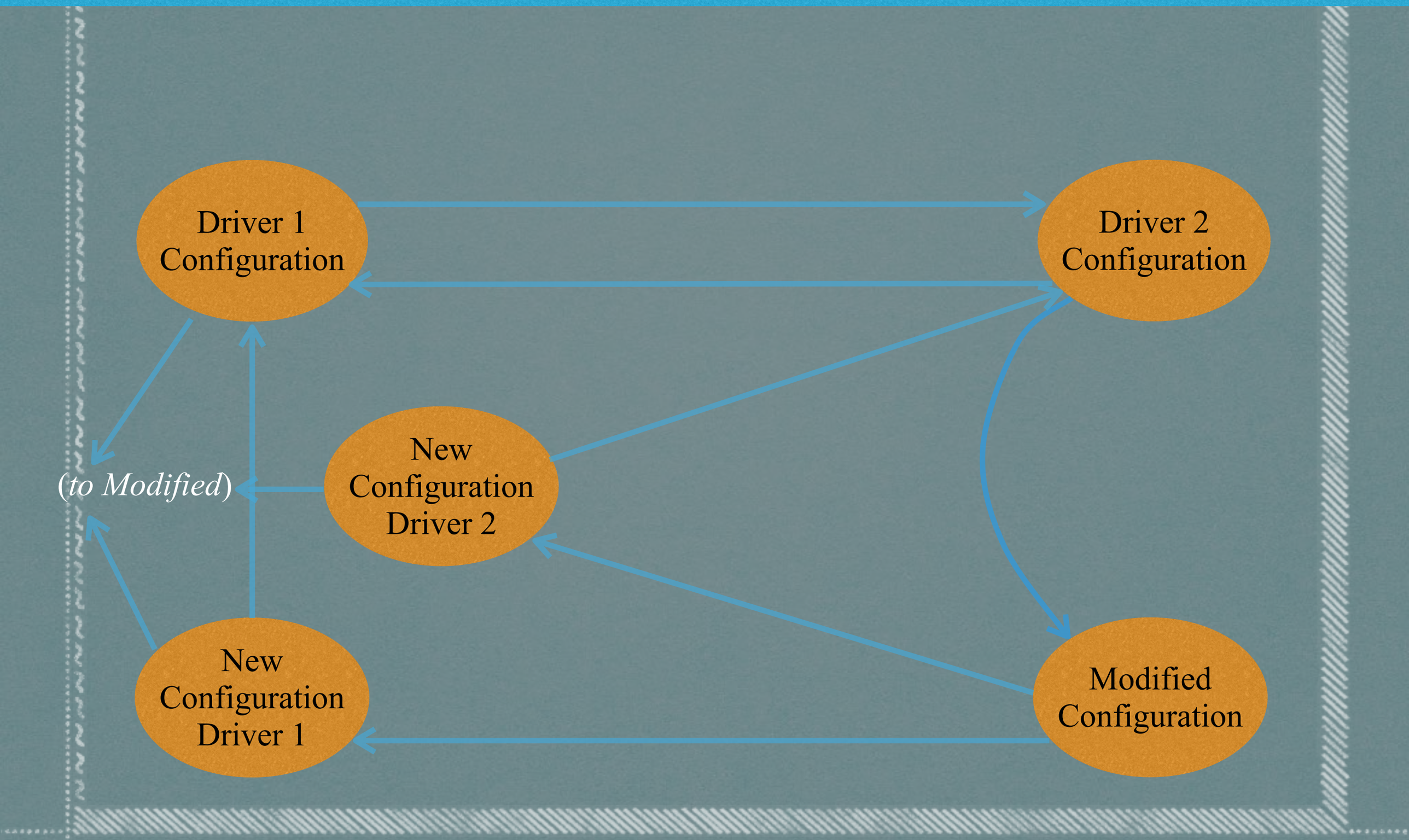
1. Graph – FSM Example
Memory Seats in a Lexus ES 350



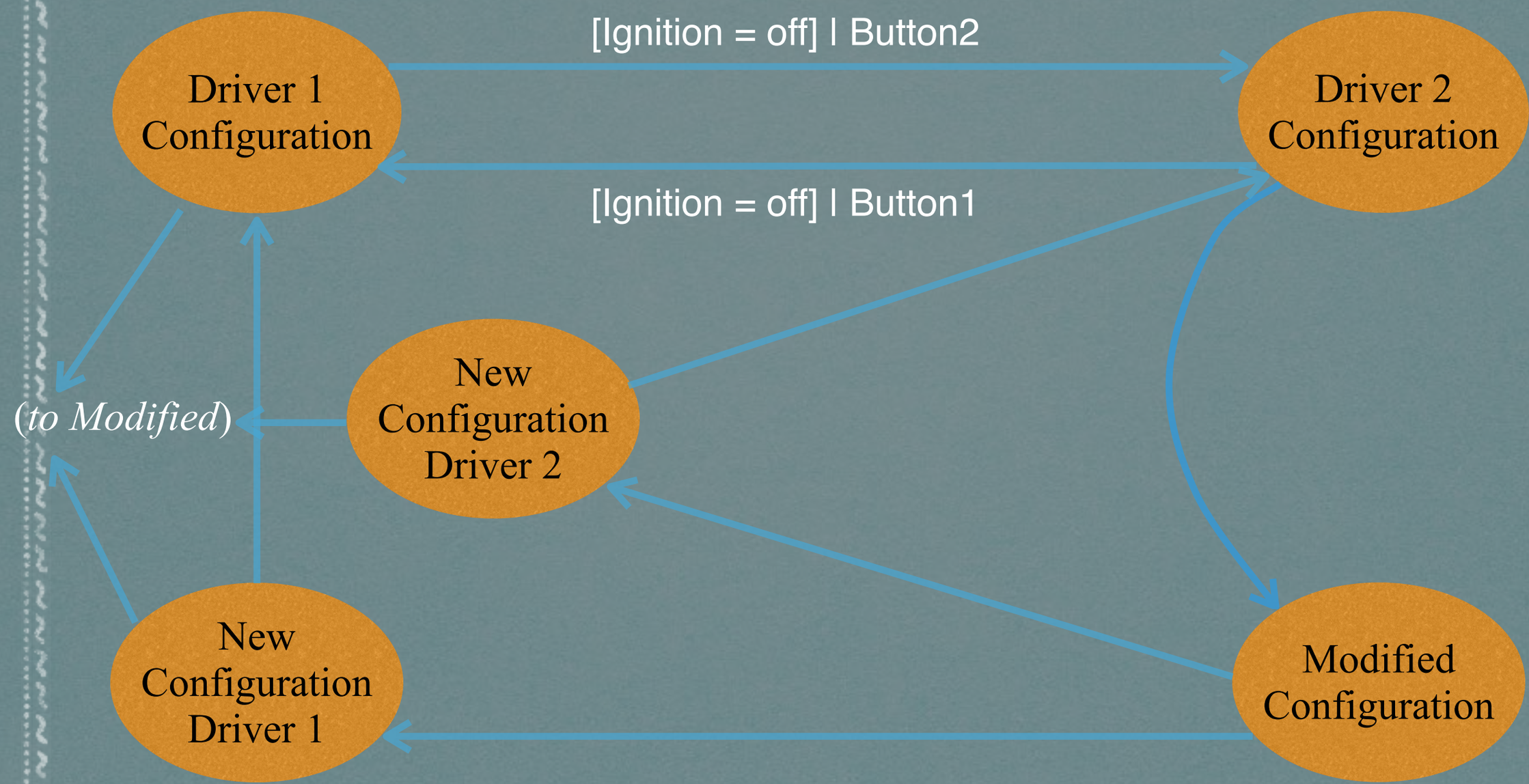
1. Graph – FSM Example
Memory Seats in a Lexus ES 350



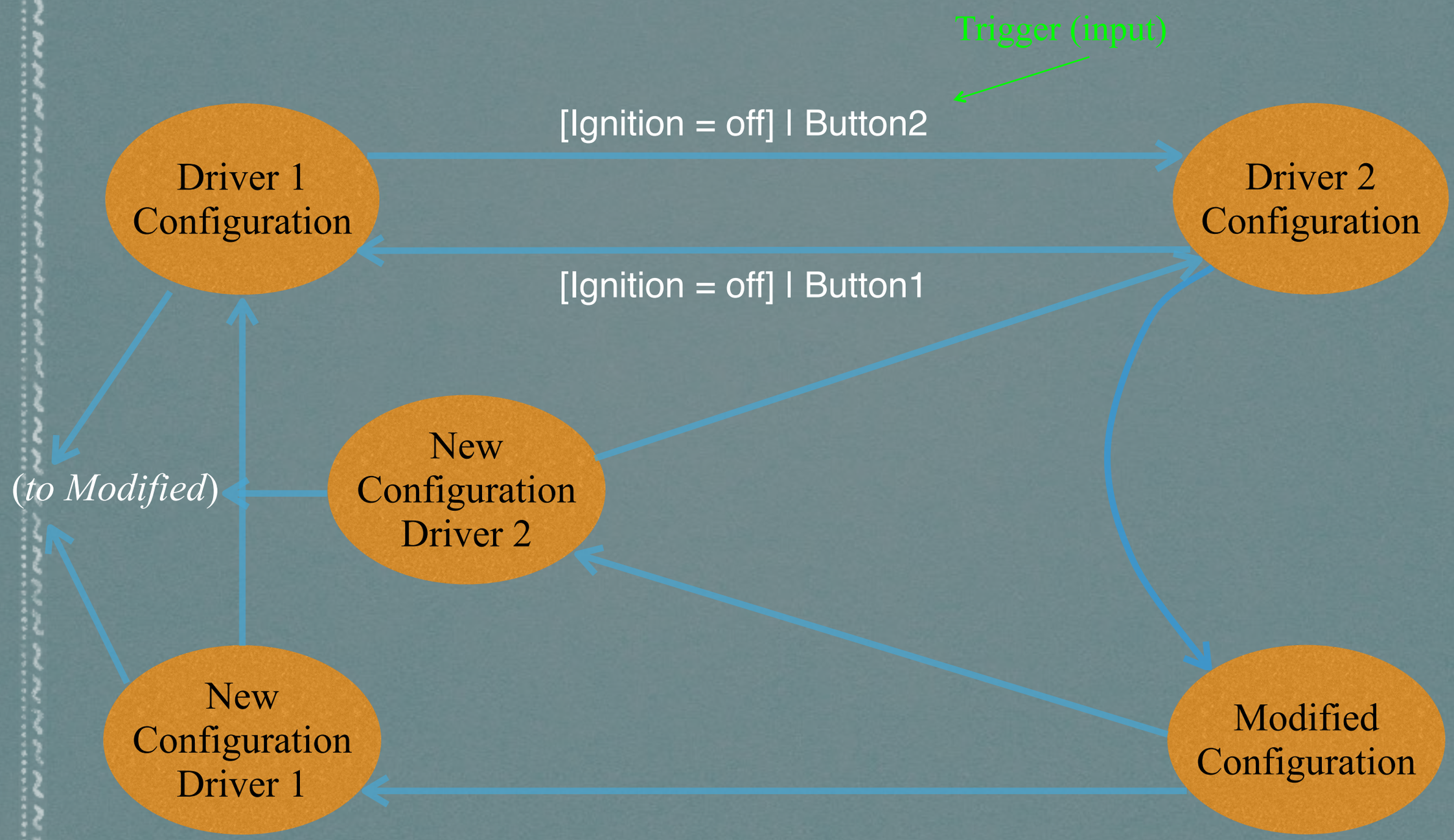
1. Graph – FSM Example
Memory Seats in a Lexus ES 350



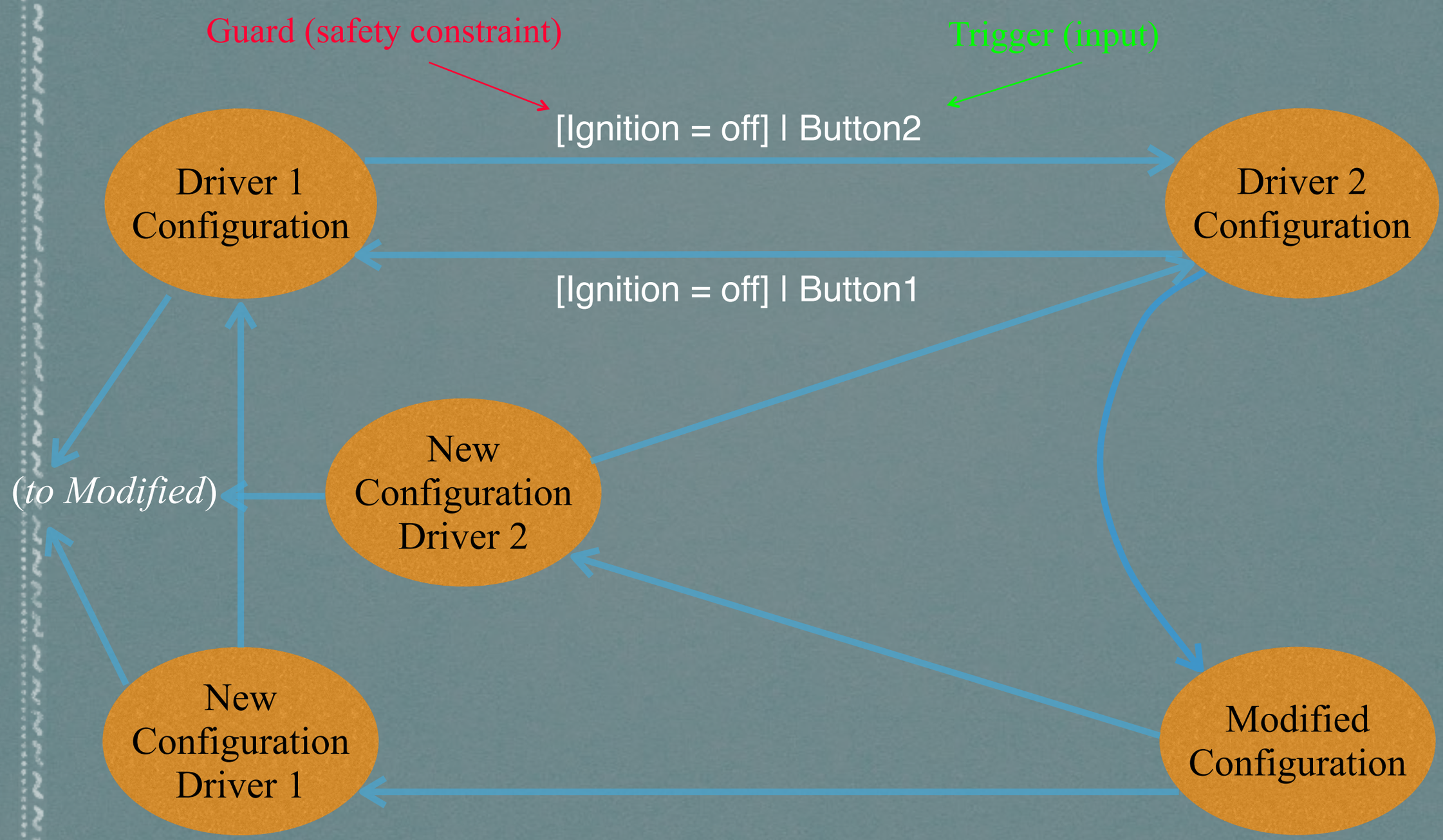
1. Graph – FSM Example
Memory Seats in a Lexus ES 350



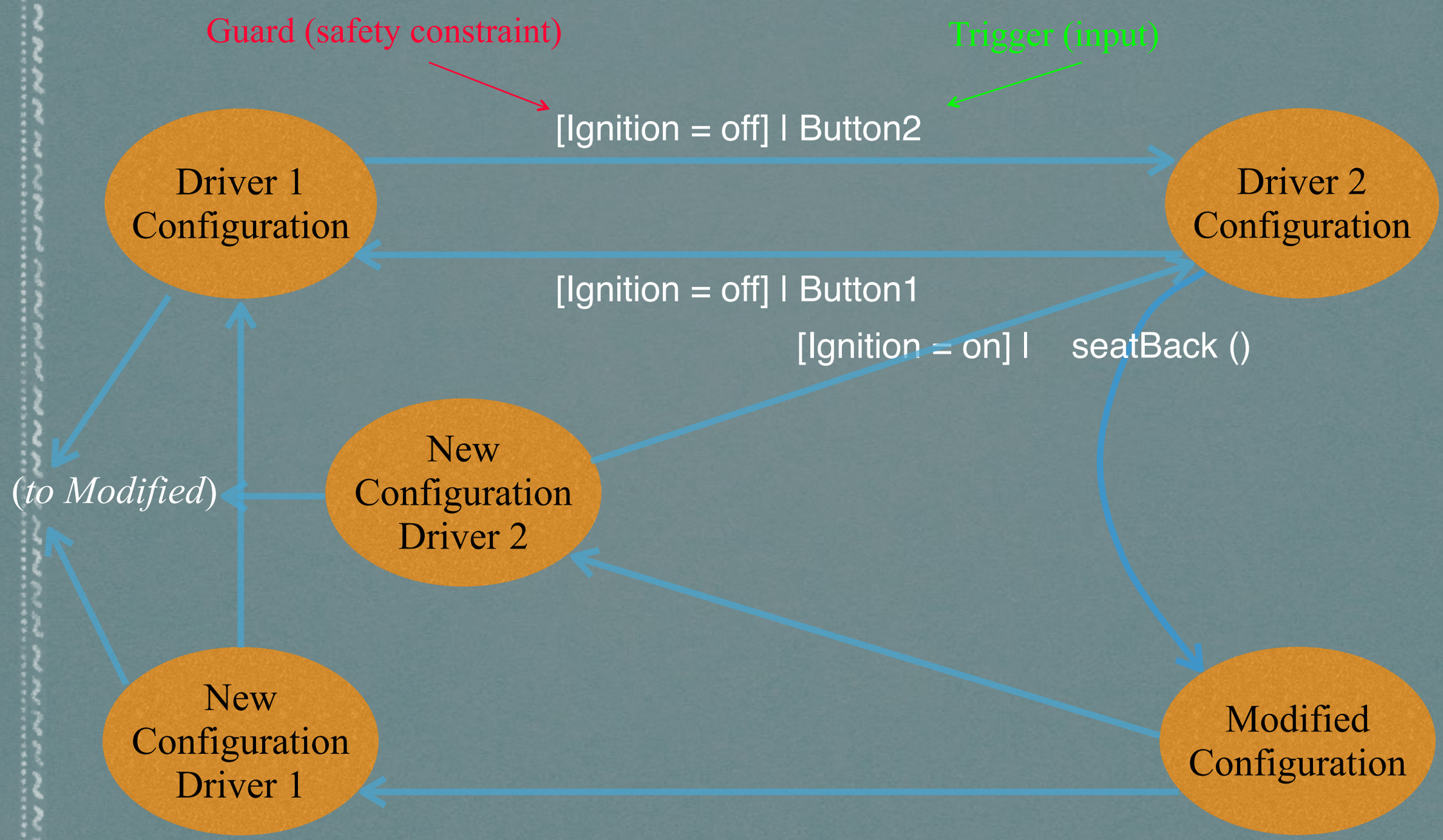
1. Graph – FSM Example
Memory Seats in a Lexus ES 350



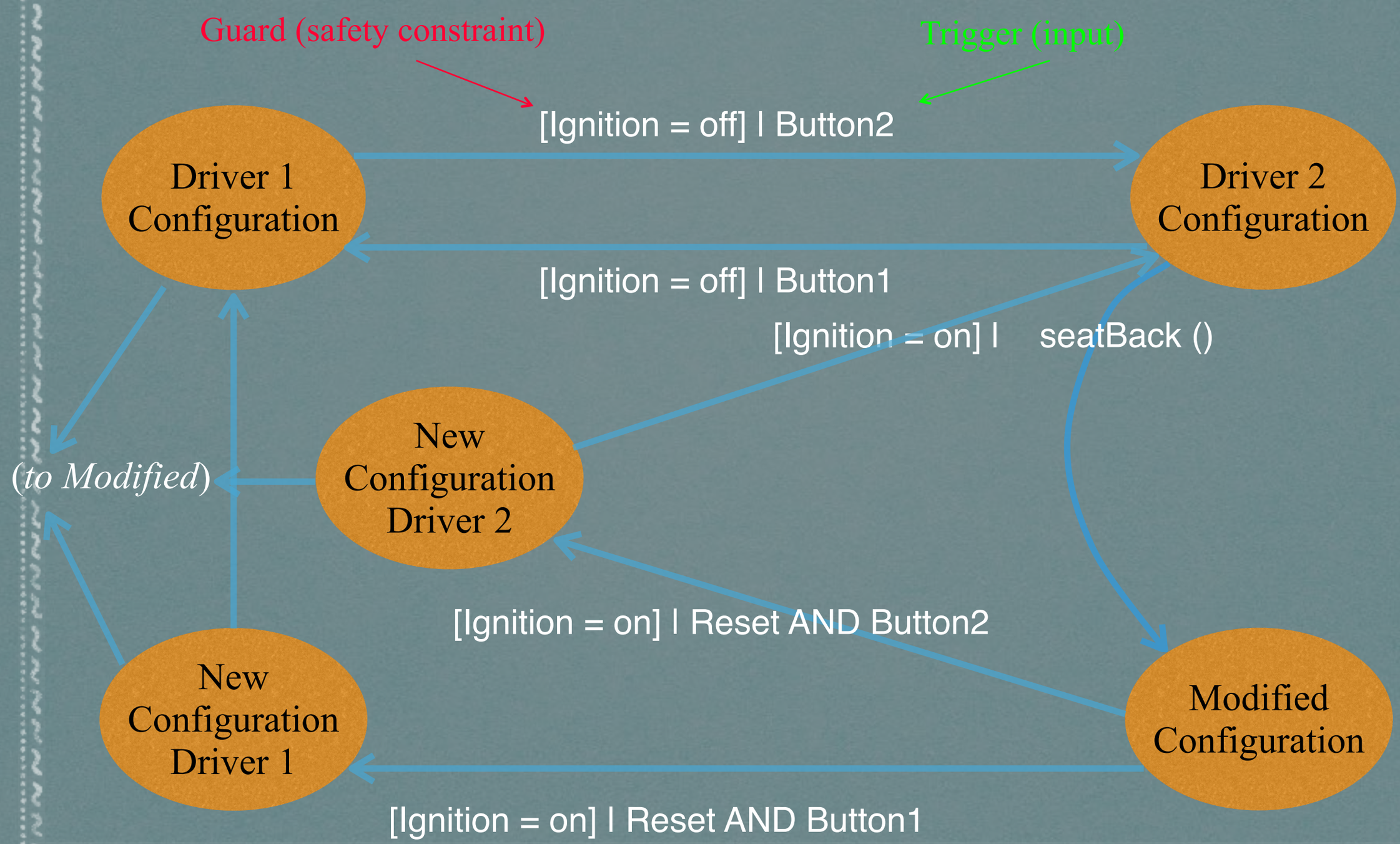
1. Graph – FSM Example
Memory Seats in a Lexus ES 350



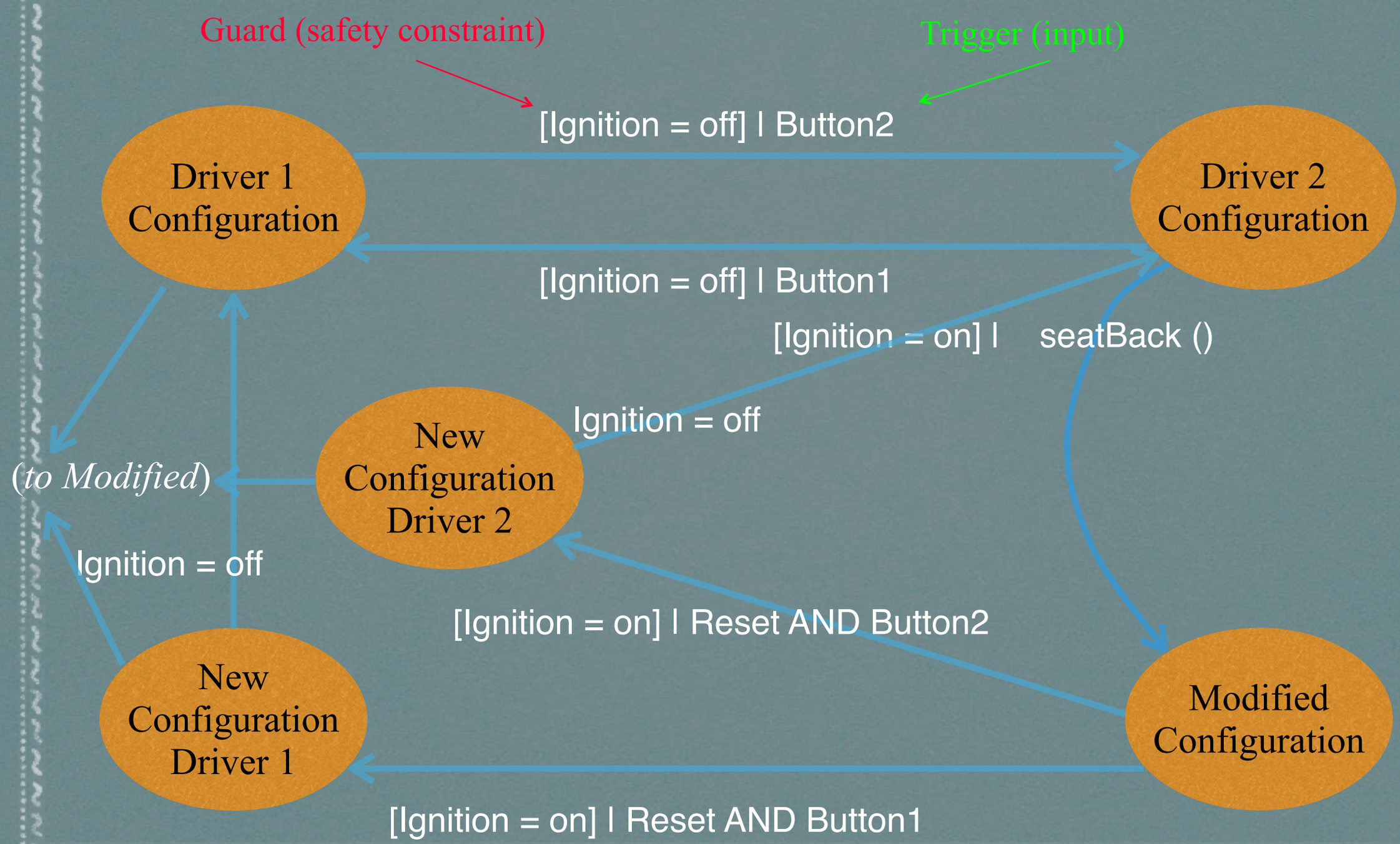
1. Graph – FSM Example
Memory Seats in a Lexus ES 350



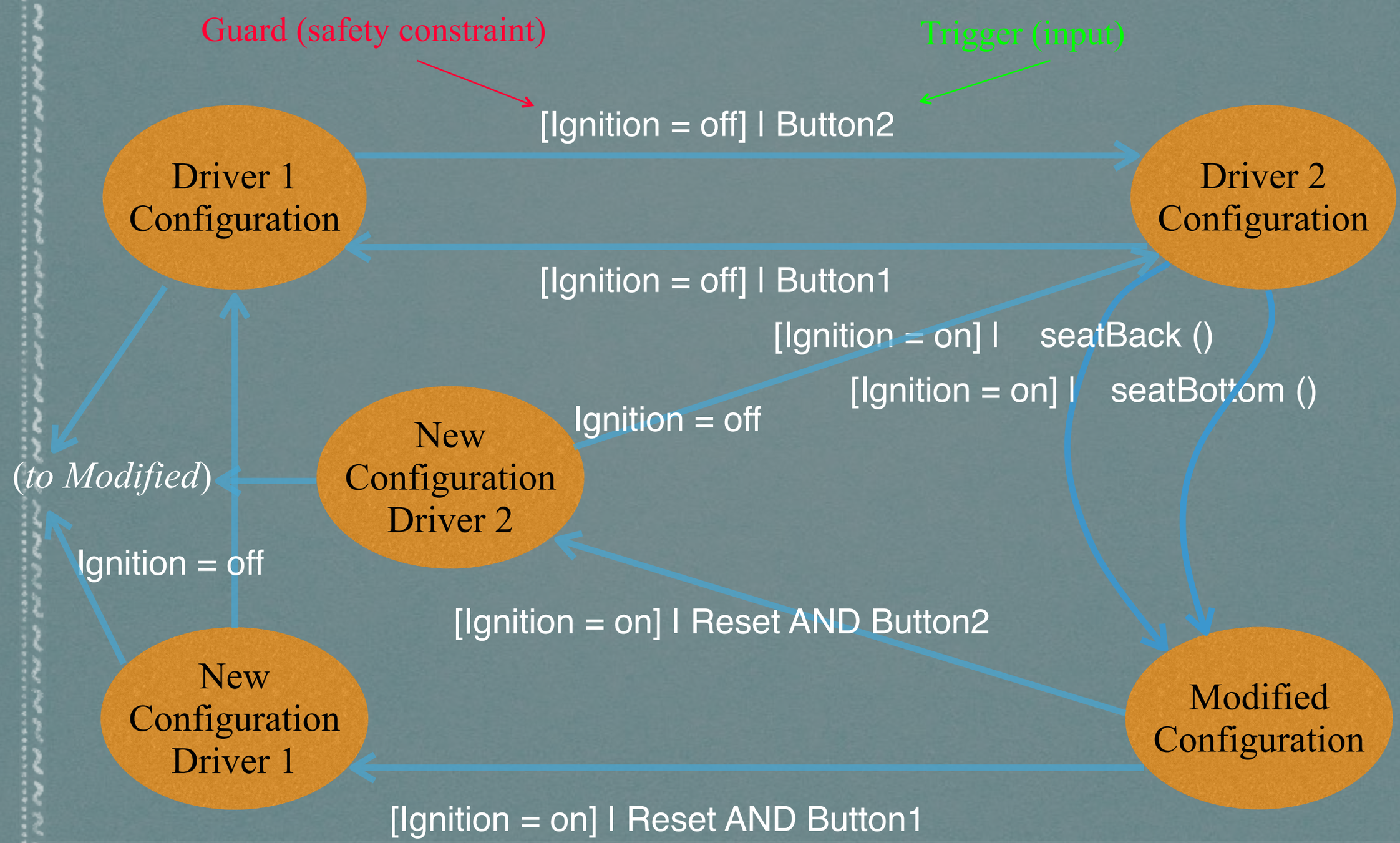
1. Graph – FSM Example
Memory Seats in a Lexus ES 350



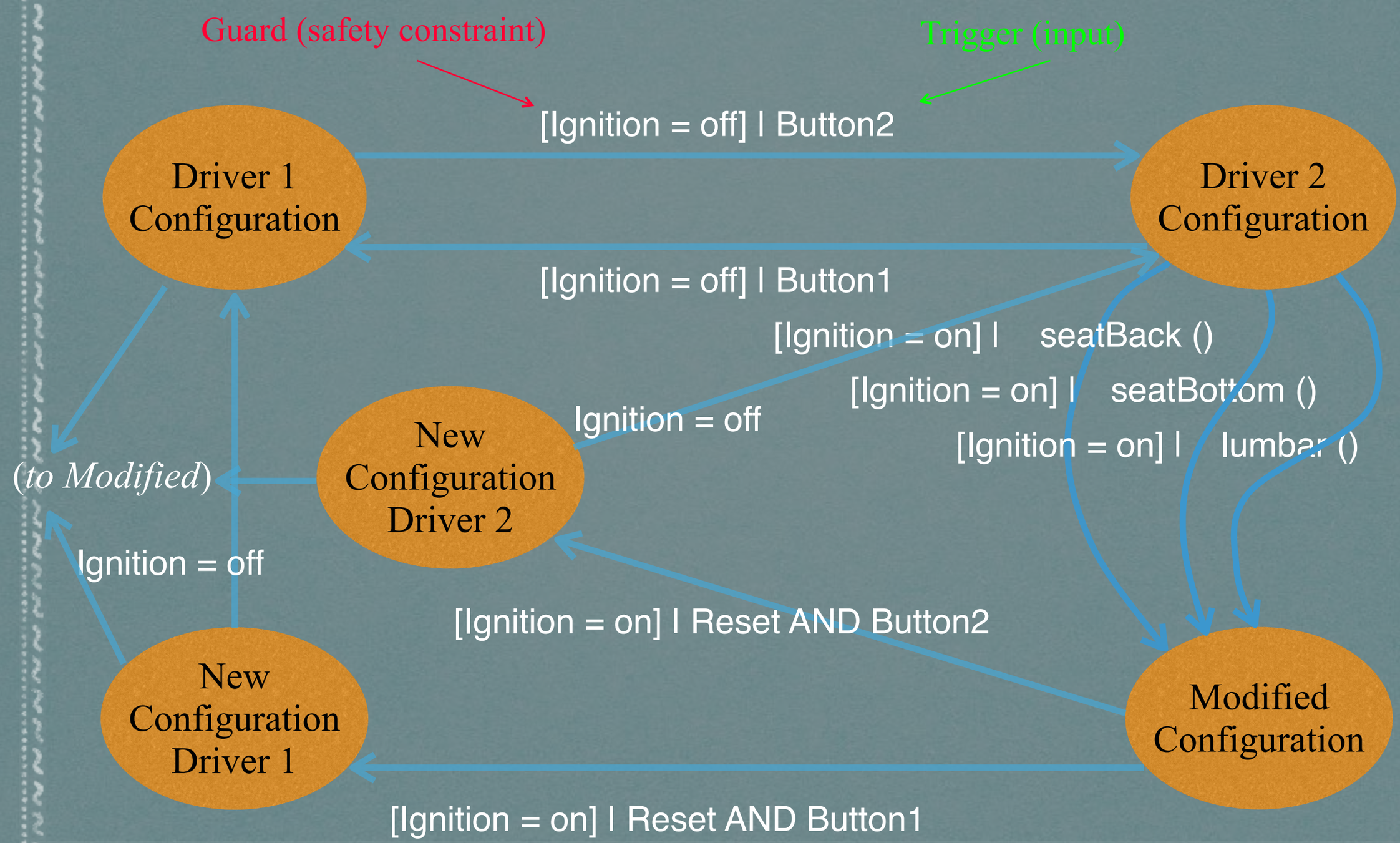
1. Graph – FSM Example
Memory Seats in a Lexus ES 350



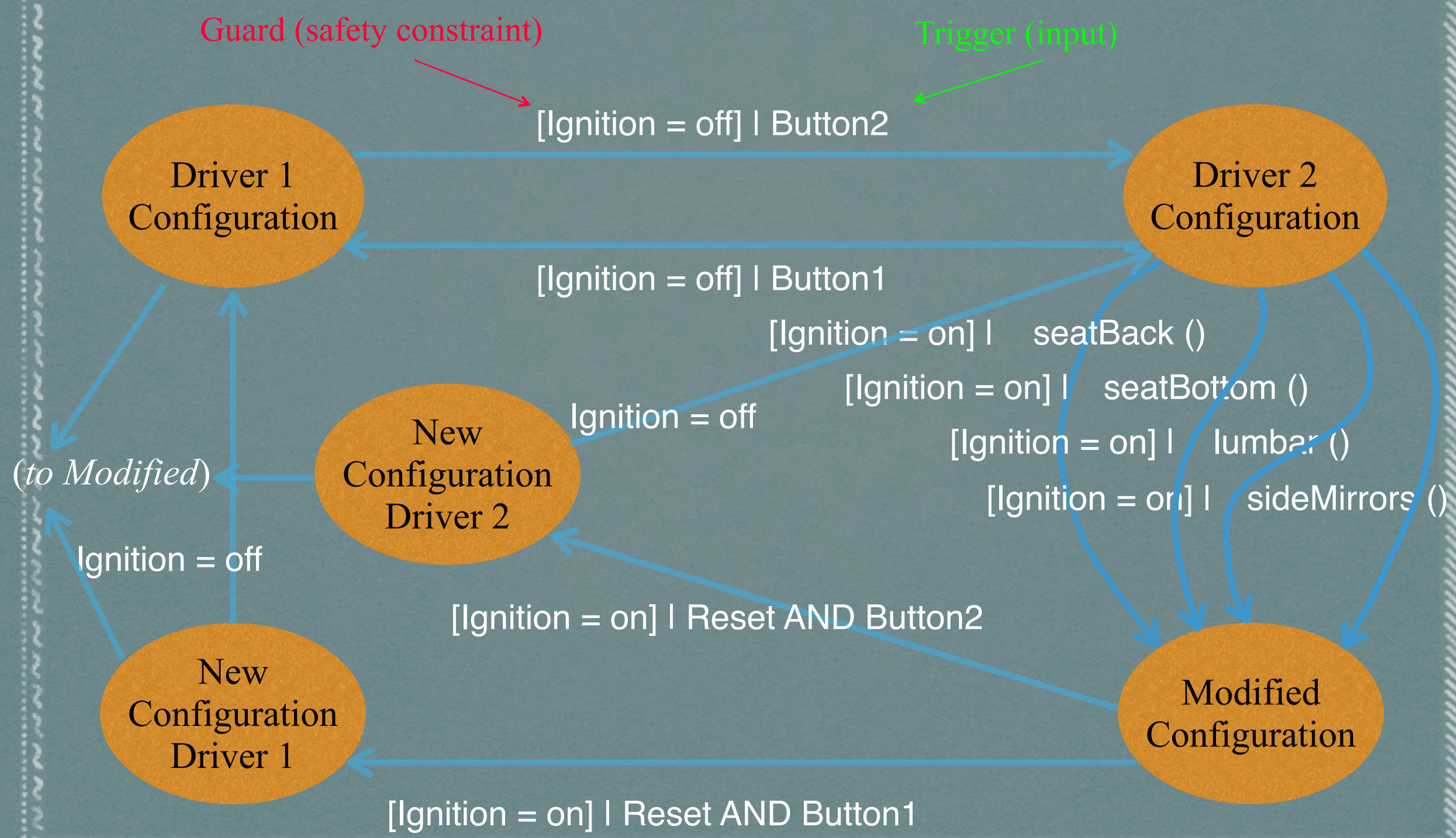
1. Graph – FSM Example
Memory Seats in a Lexus ES 350



1. Graph – FSM Example
Memory Seats in a Lexus ES 350



1. Graph – FSM Example
Memory Seats in a Lexus ES 350



2. Logical Expressions

$$((a > b) \text{ or } G) \text{ and } (x < y)$$

Transitions

Program Decision Statements

Software Specifications

2. Logical Expressions

$$((a > b) \text{ or } G) \text{ and } (x < y)$$

Transitions

Program Decision Statements

Software Specifications

Logical Expressions

2. Logical Expressions

$$((a > b) \text{ or } G) \text{ and } (x < y)$$

- Predicate Coverage : Each predicate must be true and false
  - $((a>b) \text{ or } G) \text{ and } (x < y)$ = True, False
- Clause Coverage : Each clause must be true and false
  - $(a > b)$ = True, False
  - G = True, False
  - $(x < y)$ = True, False
- Combinatorial Coverage : Various combinations of clauses
  - *Active Clause Coverage*: Each clause must determine the predicate's result

2. Logic – Active Clause Coverage

((a > b) or G) and (x < y)

1	T	F	T
2	F	F	T

2. Logic – Active Clause Coverage

With these values for G and (x<y), (a>b) determines the value of the predicate

	((a > b) or G)	and (x < y)
1	T	F	T
2	F	F	T

2. Logic – Active Clause Coverage

With these values for G and (x<y), (a>b) determines the value of the predicate

((a > b) or G) and (x < y)

1	T	F	T
2	F	F	T
3	F	T	T
4	F	F	T

2. Logic – Active Clause Coverage

With these values for G and (x<y), (a>b) determines the value of the predicate

((a > b) or G) and (x < y)

1	T	F	T
2	F	F	T
3	F	T	T
4	F	F	T
5	T	T	T
6	T	T	F

2. Logic – Active Clause Coverage

With these values for G and (x<y), (a>b) determines the value of the predicate

((a > b) or G) and (x < y)

1	T	F	T
2	F	F	T
3	F	T	T
4	F	F	T
5	T	T	T
6	T	T	F

duplicate

3. Input Domain Characterization

- Describe the input domain of the software
  - Identify inputs, parameters, or other categorization
  - Partition each input into finite sets of representative values
  - Choose combinations of values
- System level
  - Number of students { *0, 1, >1* }
  - Level of course { *600, 700, 800* }
  - Major { *swe, cs, isa, infs* }
- Unit level
  - Parameters *F (int X, int Y)*
  - Possible values X*: { <0, 0, 1, 2, >2 }, Y : { 10, 20, 30 }*
  - Tests
    - *F (-5, 10), F (0, 20), F (1, 30), F (2, 10), F (5, 20)*

4. Syntactic Structures

- Based on a grammar, or other syntactic definition
- Primary example is **mutation testing**
  1. Induce small changes to the program: **mutants**
  2. Find tests that cause the mutant programs to fail: **killing mutants**
  3. Failure is defined as different output from the original program
  4. Check the output of useful tests on the original program
- Example program and mutants

4. Syntactic Structures

- Based on a grammar, or other syntactic definition
- Primary example is **mutation testing**
  1. Induce small changes to the program: **mutants**
  2. Find tests that cause the mutant programs to fail: **killing mutants**
  3. Failure is defined as different output from the original program
  4. Check the output of useful tests on the original program
- Example program and mutants

```
if (x > y)
  z = x - y;
else
  z = 2 * x;
```

4. Syntactic Structures

- Based on a grammar, or other syntactic definition
- Primary example is **mutation testing**
  1. Induce small changes to the program: **mutants**
  2. Find tests that cause the mutant programs to fail: **killing mutants**
  3. Failure is defined as different output from the original program
  4. Check the output of useful tests on the original program
- Example program and mutants

```
if (x > y)
  z = x - y;
else
  z = 2 * x;
```

```
if (x > y)
Δif (x >= y)
  z = x - y;
Δ z = x + y;
Δ z = x – m;
else
  z = 2 * x;
```

منبع ساختار

- ساختار می‌تواند از مصنوعات مختلف نرم‌افزاری استخراج شود
  - **گراف**: از UML ، موارد کاربرد، معماری نرم‌افزار، کد برنامه
  - **عبارات منطقی**: از کد برنامه، شرط‌های موارد کاربرد، assertionها و ...

- *Model-based testing* انجام آزمون بر اساس یک مدل که از روی بخشی از سیستم تحت آزمون به دست آمده است:
  - مدل معمولا بخشی از رفتار سیستم را توصیف می‌کند.
  - کد برنامه معمولا مدل محسوب نمی‌شود.

پایان مقدمه