# Structural Equation Modelling in R

**Tom Booth**
tom.booth@ed.ac.uk

18th Jan. 2017

# Today's talk

- Introduction to SEM and a brief history
  - Logic of SEM modelling
  - What can we do with SEM?
- SEM in R
  - Package options
  - Intro to `lavaan`
- Coding SEM in `lavaan`
  - Path models, Measurement models, Full structural models
  - Parameter labels and constraints
- Running models and the core output
  - Useful wrappers
- Multi-group models
- Limits of `lavaan` and alternatives

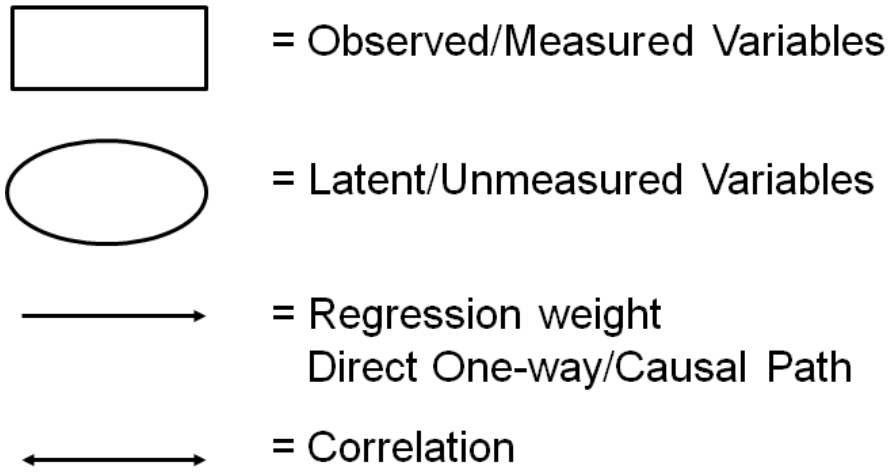
# Introduction to SEM and a brief history

# History of SEM

- SEM developed from the work of many individuals, in many fields, combining many statistical traditions;
  - Regression (Galton - biology)
  - Factor Analysis (Spearman & Thurstone - Psychology)
  - Path Modelling (Wright - Genetics)
  - Simultaneous Equations (Koopmans - Econometrics)
  - Maximum likelihood estimation (Fisher - Biology).
- Early work of Joreskog was highly influential in combining these traditions.

# Logic of SEM

- A SEM combines two broad components:
  - A **measurement model** - associations between observed and latent variables.
  - A **structural model** - associations between latent variables.
- Easiest to depict models as diagrams with certain conventions:

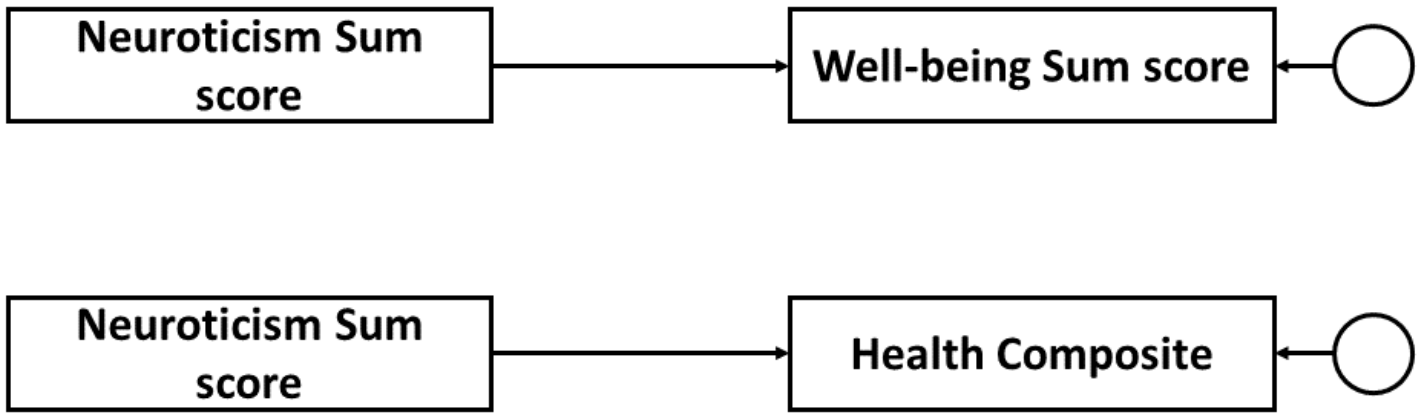


# Logic of SEM and it's advantages

- Suppose we are interested in how Neuroticism predicts psychological well-being and physical health outcomes.
  - Neuroticism measured by a questionnaire with 5 items (5-point scale).
  - Well-being is measured by a questionnaire with 5 items (7-point scale).
  - Physical health is measured based on BMI, V02 max, and presence or absence of cancer (binary).
- How do we test our model?

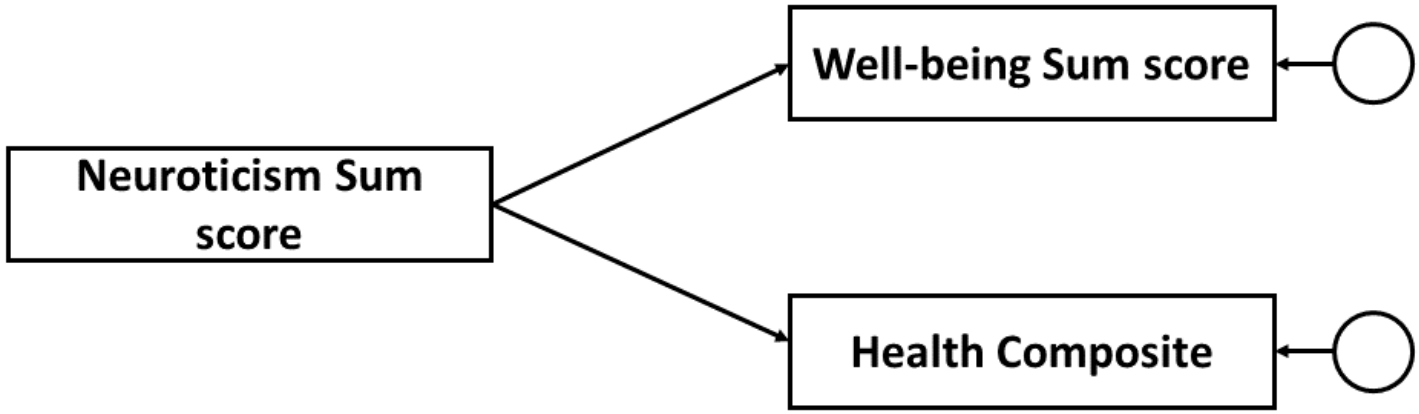
# Logic of SEM and it's advantages

- **Approach 1**: Aggregate everything into composite scores and use 2 regression models.



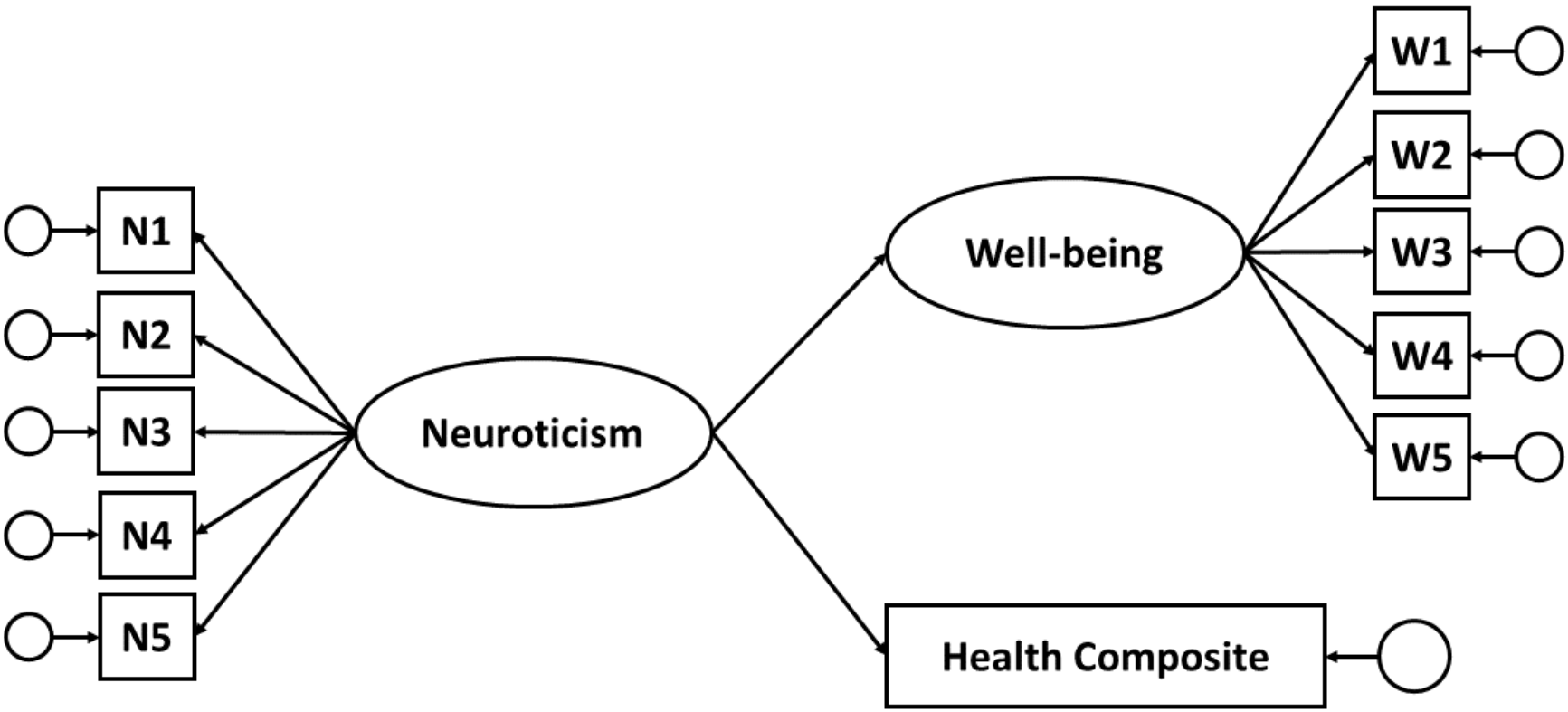
# Logic of SEM and it's advantages

- **Approach 2**: Aggregate everything, and use a path model to simultaneously estimate model with 2 outcomes.



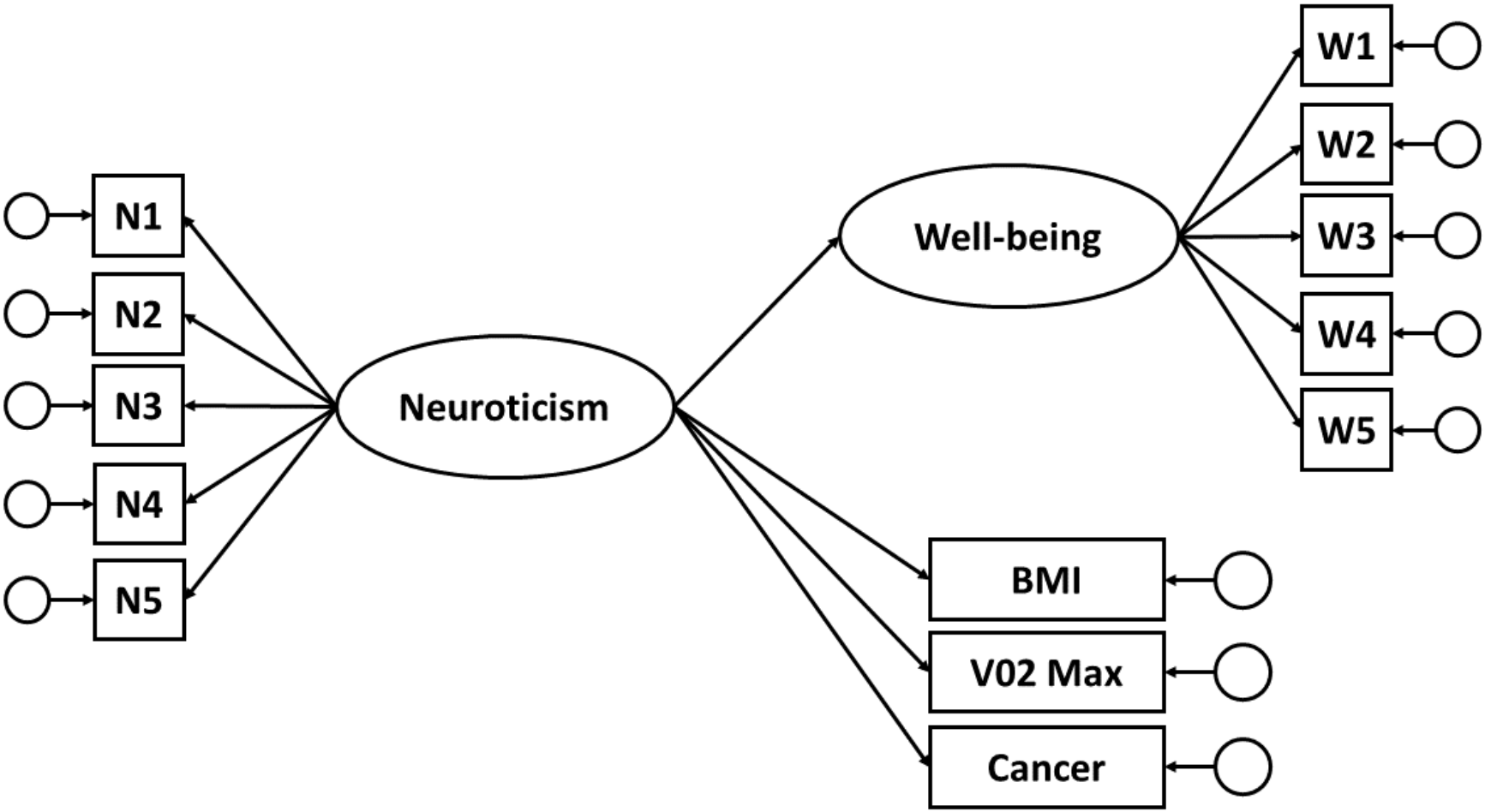
# Logic of SEM and it's advantages

- **Approach 3**: Use a mix of latent and composite variables to simultaneously estimate model with 2 outcomes.



# Logic of SEM and it's advantages

- **Approach 4**: Splitting the binary variable out.



# SEM is confirmatory

- Important to remember, **Parameters = Hypotheses**
- With each arrow(path) in a SEM model/diagram, the researcher is making a statement (hypothesis) about the associations between variables.
- Perhaps more importantly, for each arrow(path) NOT included, the researcher is also making a statement or hypothesis.

# What can we do with SEM?

- Pretty much anything!
- But we have limitations
  - Sample size
  - Estimators - particular issue for use in R which we will come back to.

# SEM in R

# Why am I talking about `lavaan`

- There are a number of packages for SEM in R.
  - `lavaan`
  - `OpenMx`
  - `sem`
- I am focussing on `lavaan` because I have always found it to be the most intuitive to use.
  - Personal preference.
- Once accustom to the core functioning, I also find the wrapper functions to be more useful than other packages.

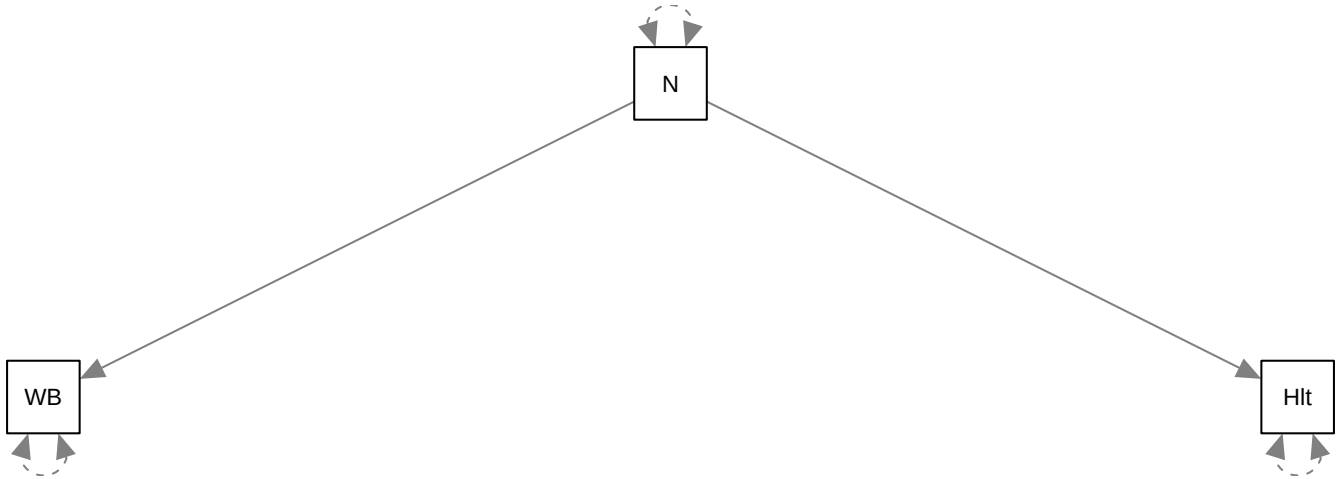
# Coding in lavaan

# Core syntax

- First job with `lavaan` is to create a model object describing the specification of the model.
  - This will need to conform to rules of SEM with respect to identification of the model.
- The basic syntax for models in `lavaan` is very simple:
  - ~ is a regression path
    - Variable to the left is DV, variable to right is IV
  - ~~ is a correlation
  - =~ is a factor loading
    - latent variable name to the left, measured variables to the right
  - + separates variables
  - For any variable which is not latent, the variable names must correspond to a variable in the data set.

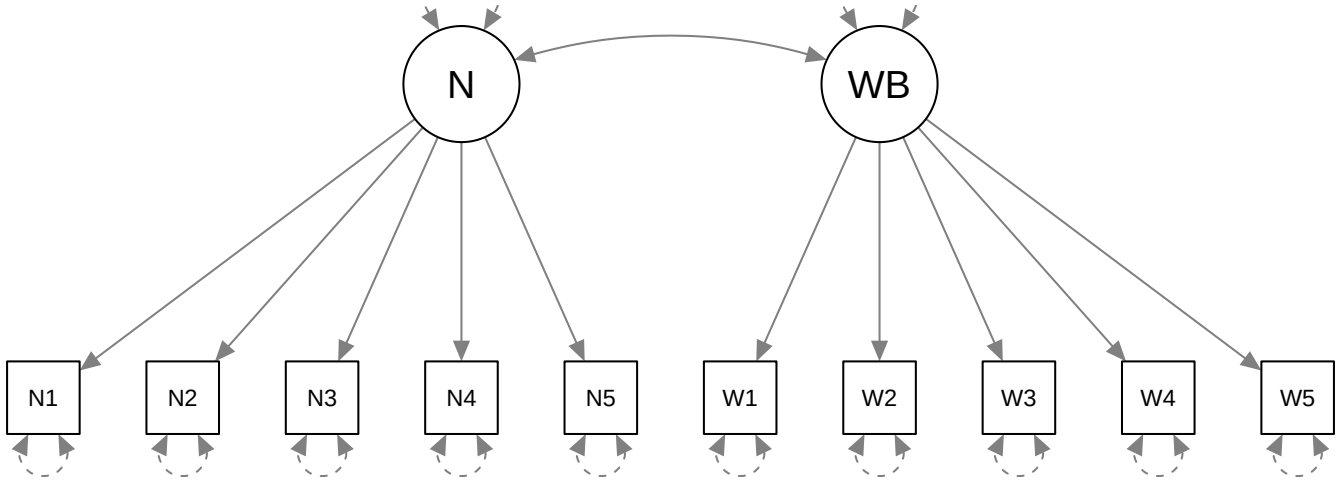
# Path models

```
pm = '
WB + Health ~ N
'
semPaths(pm)
```



# Measurement models

```
mm = '
N =~ N1 + N2 + N3 + N4 + N5
WB =~ W1 + W2 + W3 + W4 + W5
N ~~ WB
'
semPaths(mm)
```

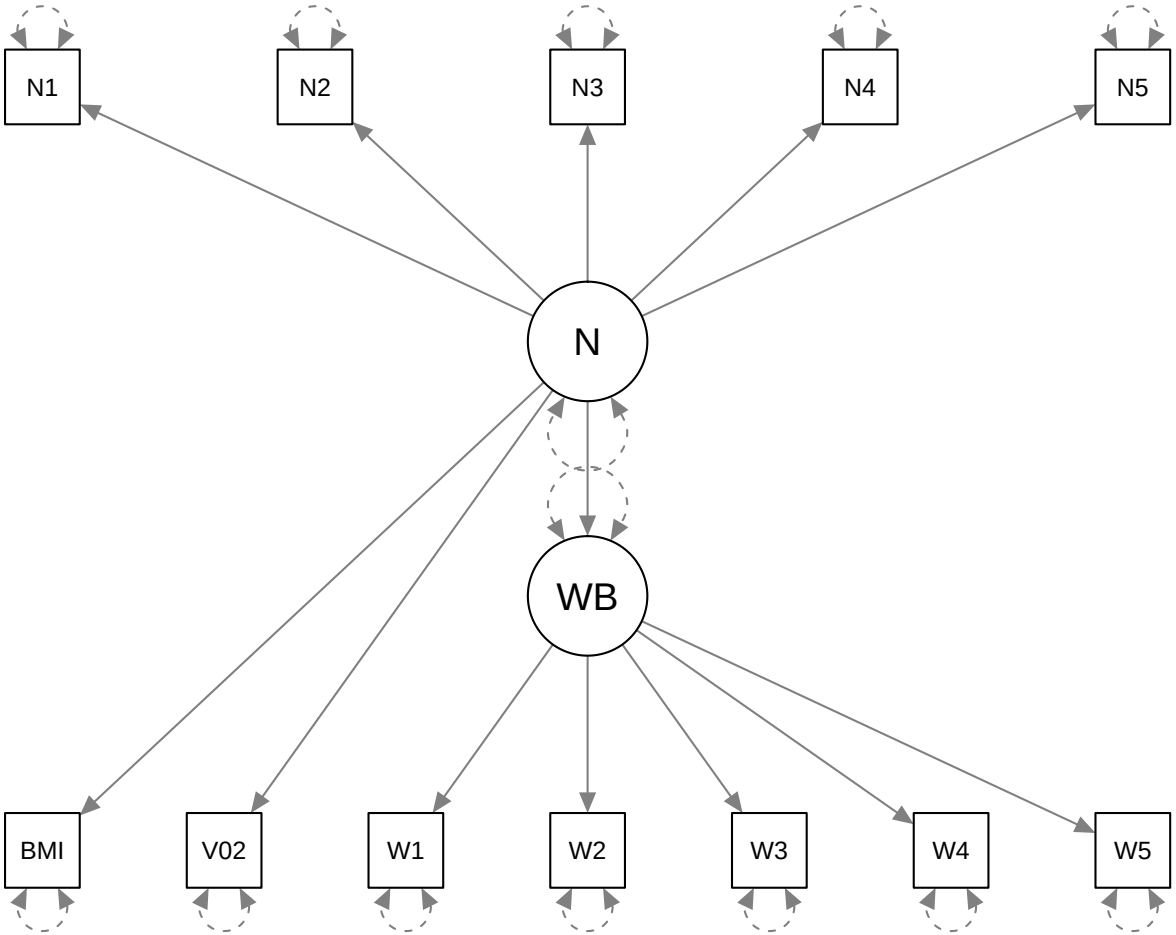


# Full structural models

```
sm = '
N =~ N1 + N2 + N3 + N4 + N5
WB =~ W1 + W2 + W3 + W4 + W5

BMI + VO2 + WB ~ N
'
```

# Full structural models

```
semPaths(sm)
```



# Parameter labels and constraints

- A couple of other useful elements to coding in SEM are parameter constraints and labels.
- Fixing a parameter to specific value uses as *

```
## N1 factor loading fixed to 1
sm = '
N =~ 1*N1 + N2 + N3 + N4 + N5
WB =~ W1 + W2 + W3 + W4 + W5

BMI + VO2 + WB ~ N
'
```

# Parameter labels and constraints

- Fixing starting values uses `start()`

```
## Fixes the start value for N1 factor loading to 0.4
sm = '
N =~ start(0.4)*N1 + N2 + N3 + N4 + N5
WB =~ W1 + W2 + W3 + W4 + W5

BMI + VO2 + WB ~ N
'
```

# Parameter labels and constraints

- Fixing parameters to be equal involves giving them the same label.

```
## Fixes the loadings of N1 and N2 to be equal
sm = '
N =~ bob*N1 + bob*N2 + N3 + N4 + N5
WB =~ W1 + W2 + W3 + W4 + W5

BMI + VO2 + WB ~ N
'
```

# Running models and output

# Model estimation

- Most open model estimation uses `lavaan()`, where not defaults are set.
- We will look at two basic examples using some of the convenience wrappers and contrast them to `lavaan()`:
  - Measurement models using `cfa()`
  - General structural model using `sem()`

# Arguments to cfa() and sem()

- Here we use the function `cfa()` within lavaan.

```
cfa(model=, data=, std.lv=, std.ov=, missing=,
    estimator=)
```

- `model=` : Here we add the name of our model object (examples below)
- `data=` : As always we need to give our data set, where variable names match the model object
- `std.lv=` : a TRUE or FALSE statement, scale the latent variable by first loading (F) or latent variance (T)

# Arguments to cfa() and sem()

```
cfa(model=, data=, std.lv=, std.ov=, missing=,
    estimator=)
```

- `std.ov=` : Standardize the observed variables - TRUE or FALSE
- `missing=` : How missing data is treated. - Big advantage of CFA/SEM is use of full information maximum likelihood (FIML) with ML estimation.
- `estimator=` : Many options by ML and DWLS are likely most useful.

# mimic

- For those coming to R from other SEM packages, `lavaan` offers a very useful `mimic =` argument
- This essentially sets up a given model (CFA or SEM) to have the same default parameter specification as;
  - MPlus
  - EQS
- Similarly, the `representation = LISREL` is specified, the model is presented in LISREL matrix notation.

# Core output

- Once a model has been run and saved as an object, there are a number of evaluation functions to call.
- As with `lm()`, `summary()` provides an overview.
  - `summary()` can also include many of the following...
- For model fit (nested and non-nested)
  - `AIC()`, `BIC()`, `fitmeasures()`
- Parameter estimates
  - `parameterEstimates()`, `standardizedSolution()`
- Modification indices
  - `modindices()`

# Some actual data

- To show the model output, we will actually fit a small SEM model to data from the Big Five Index (BFI) measure of personality (psych package)
- So we can see the various matrices, we will fit a model that predicts Conscientiousness from age in years.
    - A silly model I know!

# The model

```
library(psych)
library(lavaan)
N_age = '
N =~ N1 + N2 + N3 + N4 + N5
N ~ age
'
test <- sem(N_age, data = bfi, estimator="ml", std.lv=T)
```

# Model fit

- All models are wrong, but some are more wrong than others.
  - But how do we assess how wrong our model is?
- In CFA and SEM, a first step in evaluating our proposed models is to assess model fit.
  - There are a huge number of fit measures, all of which have the general aim of evaluating how well our model fits our data.
- Once we have assessed the fit of our model, we can consider the significance and effect sizes of our parameter estimates.

# Model fit measures

| Fit Index | Suggested Cut-offs | Parsimony Correction | Compare non-nested models? |
|---|---|---|---|
| $X^2$ | $p < 0.05$ | No | No |
| $X^2$/df ratio | 3:1 (ish!) | $X^2$/df ratio | No |
| CFI | >0.90 to 0.95 | 1 per estimated param | No |
| TLI | >0.90 to 0.95 | $X^2$/df ratio | No |
| RMSEA | <0.05 to 0.08 | $X^2$/df ratio | No |
| SRMR | <0.05 to 0.08 | None. | No |
| AIC | Smaller the better | 2*k | Yes |
| BIC | Smaller the better | Log(N)*k | Yes |
| saBIC | Smaller the better | Log((N+2)/24)*k | Yes |

# View model fit

```
fitMeasures(test, c("rmsea", "srmr", "cfi", "tli"))
```

```
## rmsea  srmr   cfi   tli
## 0.124 0.049 0.922 0.870
```

- Some of these dont look so great. . .

# Modification indices

```
modindices(test)
```

```
##     lhs op rhs      mi    epc sepc.lv sepc.all sepc.nox
## 13  age ~~ age   0.000  0.000   0.000    0.000    0.000
## 14   N1 ~~  N2 389.235  0.821   0.821    0.342    0.342
## 15   N1 ~~  N3  58.845 -0.291  -0.291   -0.115   -0.115
## 16   N1 ~~  N4  70.397 -0.277  -0.277   -0.112   -0.112
## 17   N1 ~~  N5  24.232 -0.168  -0.168   -0.066   -0.066
## 18   N2 ~~  N3  50.181 -0.256  -0.256   -0.105   -0.105
## 19   N2 ~~  N4  62.016 -0.252  -0.252   -0.105   -0.105
## 20   N2 ~~  N5  48.126 -0.229  -0.229   -0.093   -0.093
## 21   N3 ~~  N4 170.143  0.443   0.443    0.176    0.176
## 22   N3 ~~  N5  46.688  0.243   0.243    0.094    0.094
## 23   N4 ~~  N5  87.067  0.358   0.358    0.141    0.141
## 24  age  ~   N   0.000  0.000   0.000    0.000    0.000
```

# Revise the model

- Not always preferable, but will allow me to show some more elements

```
library(psych)
library(lavaan)
N_age2 = '
N =~ N1 + N2 + N3 + N4 + N5
N ~ age

## modification
# N1 = get angry easily
# N2 = get irritated easily
N1 ~~ N2
'
test2 <- sem(N_age2, data = bfi, estimator="ml", std.lv=T)
```

# Compare model fit

```
fitMeasures(test, c("rmsea", "srmr", "cfi", "tli"))
```

```
## rmsea  srmr   cfi   tli
## 0.124 0.049 0.922 0.870
```

```
fitMeasures(test2, c("rmsea", "srmr", "cfi", "tli"))
```

```
## rmsea  srmr   cfi   tli
## 0.044 0.020 0.991 0.983
```

# Parameter estimates

```
head(standardizedSolution(test2))
```

```
##   lhs op rhs est.std    se      z pvalue
## 1   N =~  N1   0.670 0.014 47.278      0
## 2   N =~  N2   0.654 0.015 44.934      0
## 3   N =~  N3   0.817 0.012 68.895      0
## 4   N =~  N4   0.631 0.015 43.464      0
## 5   N =~  N5   0.551 0.016 34.400      0
## 6   N  ~ age  -0.128 0.021 -6.085      0
```

# Useful wrappers/functions

- There are also a set of on-going related projects which integrate well:
  - `lavaan.survey` for complex data structures.
  - `Onyx` provides a graphical interface
  - `semPlot` produces diagrams
  - `semTools` contains LOTs of useful functions for all things SEM.
    - `compareFit()`
    - `measurementInvarianve()`, `measurementInvarianceCat()`
    - `parcelAllocation()`
    - `reliability()`
  - `simsem` for conducting simulations with SEM

# Multi-group model

# General multiple group analysis

- Multiple group analysis is one of the coolest (in my opinion) features of SEM.
- It makes it very easy to test equality constraints in any type of model.
- In essence our model is fit in both groups, and we can fix values to be the same, different, conform to some rule, in our different groups.
- One particularly important example of these models in psychology (and latent variable modelling) is measurement invariance.

# Coding multi-group models

- Very simple. We simply add `group = [variable name]` to our model run.

```
Neuro = '
WB + Health ~ N
'

sem(Neuro, data=data, std,lv=T, estimator="ml",
    group="country")
```

# Adding equality constraints

- By default this fits whatever model is given in all groups, and freely estimates the parameters in each group.
- To add constraints, we simply use the same code format as before, but provide a vector of names.
- So (assuming a three group model)...

# Adding equality constraints

- This model would fix the regression coefficient for neuroticism and health equal across groups:

```
Neuro = '
WB ~ N
Health ~ a*N
'

cfa(Neuro, data=data, std,lv=T, estimator="ml",
    group="country")
```

# Adding other constraints

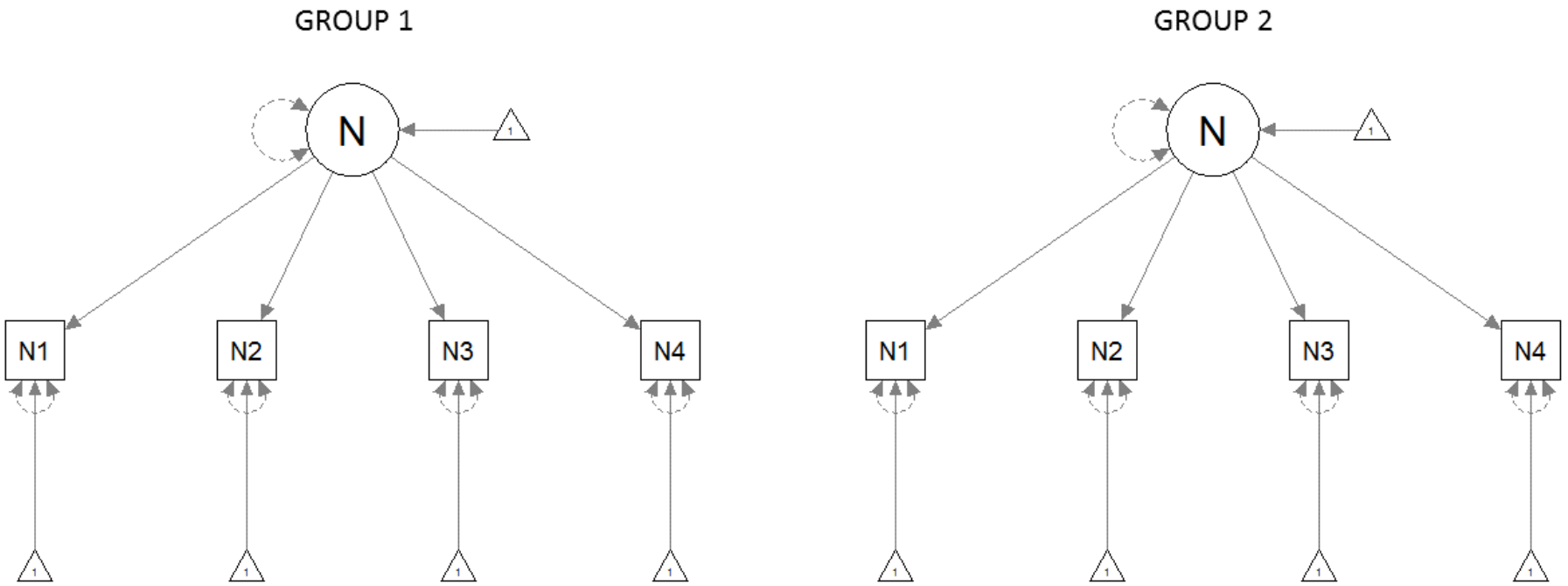
- Whereas this would label each differently, and we could then add constraints:

```
Neuro = '
WB ~ N
Health ~ c(a1, a2, a3)*N

# Constraints
a1 > a2
a2 > a3
'

cfa(Neuro, data=data, std,lv=T, estimator="ml",
    group="country")
```

# MG-CFA: Measurement Invariance

- Measurement invariance assess whether latent variables are equivalent across groups by constraining different aspects of the measurement model.
    1. Pattern factor loadings: **configural invariance**
    2. Magnitude of factor loadings: **metric invariance**
    3. Item intercepts: **scalar invariance**
    4. Residual variances: **strict invariance**
- The labels differ a little across papers/discussions, but the levels are the same.
- Different questions may require different levels of invariance.

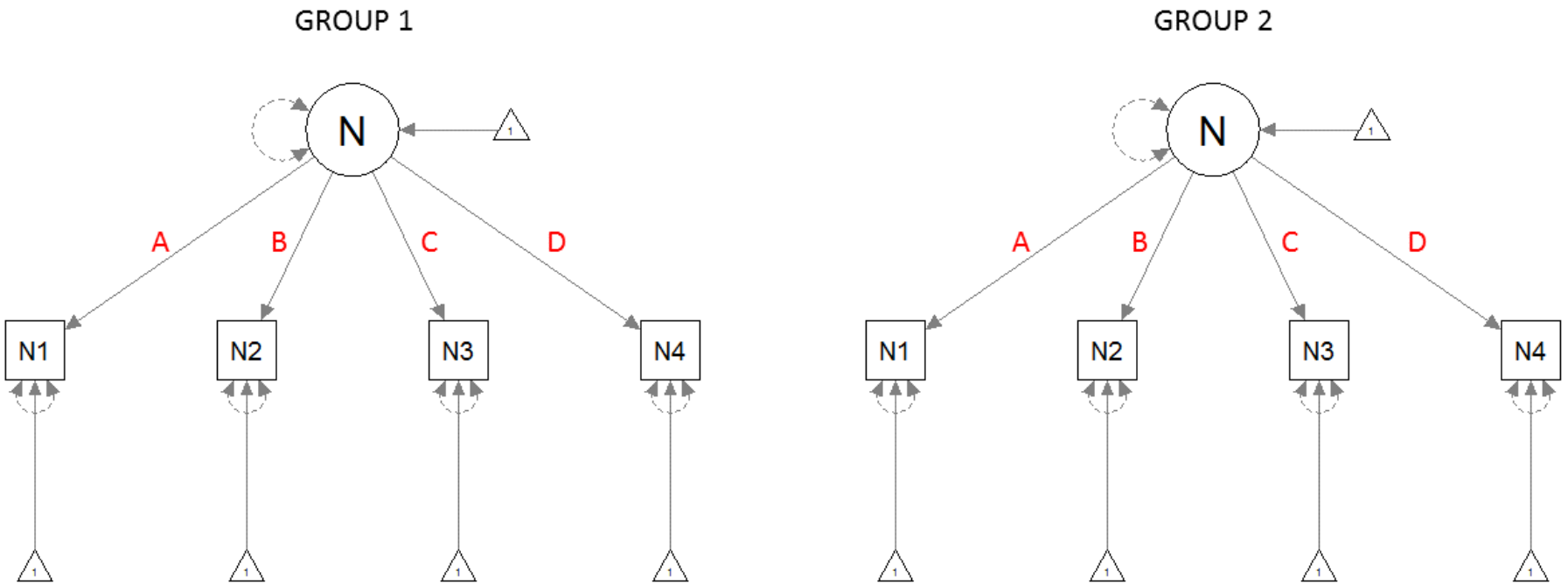
# MG-CFA: Configural Invariance (1)



# MG-CFA: Configural Invariance (2)

- What we are testing:
  1. That the number of factors is identical in both groups.
  2. That the non-zero elements in $\Lambda_k$ are identical.
     - This is only a concern with more than1 factor.
- Identification:
  - Fix a factor loading in both groups to 1.
    - Issue here, is this item invariant?
  - Constain the factor means to 0 in one group
- Note big difference here is we need to think about identification of our factor model, and our means structure.

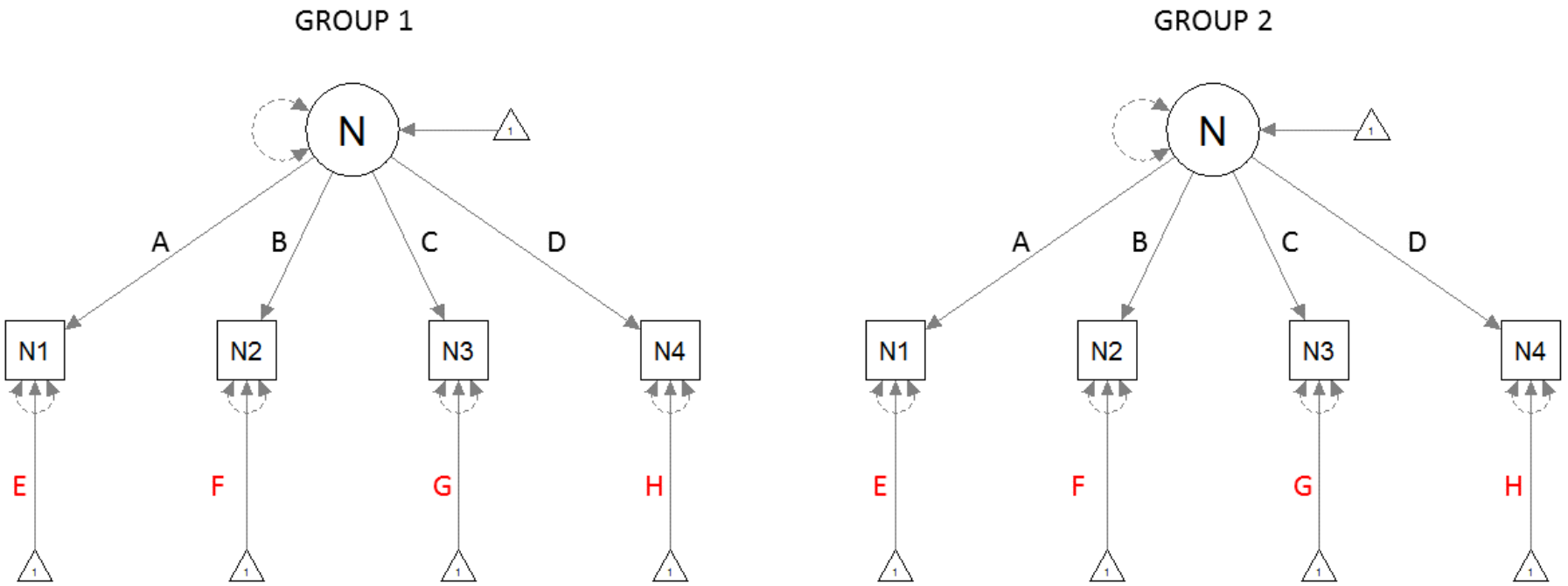
# MG-CFA: Metric Invariance (1)



# MG-CFA: Metric Invariance (2)

- What we are testing:
  1. That the magnitude of the estimates of the factor loadings are equal across groups.
     - Metric invariance helps us to ensure the interpretation of the factors is the same across groups.
     - Metric invariance is necessary but not sufficient here.
- Identification:
  - We retain our identification constraints from the previous model.
- **Note**: There are many different options for identification when conducitng invariance analyses, here is just one set.

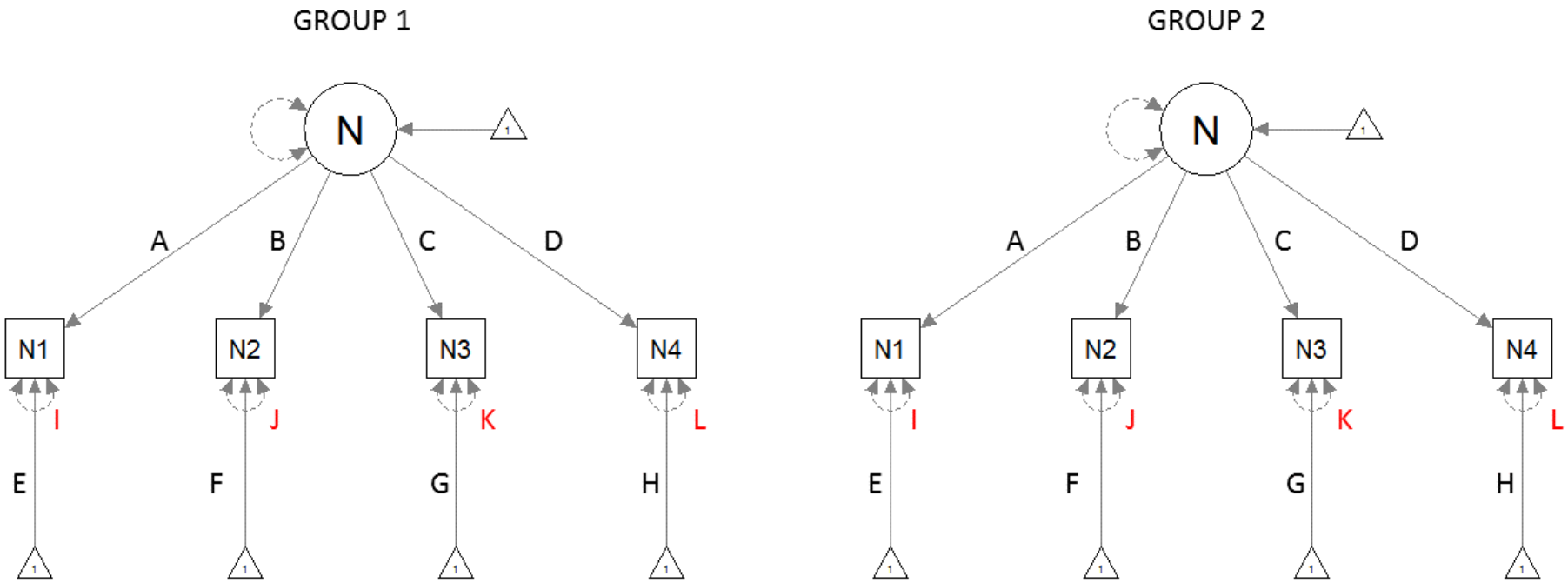
# MG-CFA: Scalar Invariance (1)



# MG-CFA: Scalar Invariance (2)

- What we are testing:
  1. That the intercepts of the items are equal across groups.
     - By this point all elements of our regression model involving the latent variable and item are fixed; the loading (or weight) and the intercept.
- Identification:
  - We retain our identification constraints from the previous model.

# MG-CFA: Strict Invariance (1)



# MG-CFA: Strict Invariance (2)

- What we are testing:
  1. That the residual variances are equal across groups.
- The residual captures true unique variance and error variance.
  - A difference across groups in true unique variance may suggest differences in, say, understanding of idiosyncractic features like item wording.
  - If the difference is error variance, then lack of invariance may suggest different item reliability across groups.
- Formally, loadings and residual invariance is needed for true interpretation of factor variances and covariances.
  - Remains some discussion concerning other models it may be required for.

# MG-CFA: Evaluating MG-CFA

- The set of measurement invariance levels provide a group of *nested models*.
- A model is said to be nested when:
  1. It uses the same variables.
  2. It uses the same sample.
  3. The models differ in the estimated parameters.
     - If one model can be specified as a more constrained version of the other.
- When we fix a parameter in a model, it is considered to be *nested* within the model with that parameter freely estimated.
- Given this, we can test the difference in fit between our models in a number of ways.

# MG-CFA: Difference in Fit Criteria

- $\chi^2$*difference* test.
  - A significant value for the test indicates the model fit is significantly worse.
- Based on Chen (2007) invariance is considered to hold if:
  - $\Delta CFI \leq -.01$
  - $\Delta RMSEA \leq .015$

# MG-CFA: Invariance in R (1)

```
# Configural Invariance
config <- cfa(model, data, group = "Sex")

# Metric
metric <- cfa(model, data, group = "Sex",
              group.equal=c("loadings"))

# Scalar
scalar <- cfa(model, data, group = "Sex",
              group.equal=c("loadings", "intercepts"))

# Strict
metric <- cfa(model, data, group = "Sex",
              group.equal=c("loadings", "intercepts",
                            "residuals"))
```

# MG-CFA: Invariance in R (2)

- The above code let's us build our invariance models sequentially.
- We could then use the `semTools` function `compareFit()` to consider the fit across models.
- We could also use the function `measurementInvariance()` from `semTools`.
    - This automates the analysis and runs sequentially more constrained models.
    - It also provides fit comparisons.

# MG-CFA: Partial Invariance (1)

- What if model fit suggests that invariance does not hold?
- As with all model building, we could apply different strategies:
  - Backwards exploration: Start with all values fixed, and gradually free parameters.
  - Forwards exploration: Start with all free and gradually constrain.
- In the context of invariance, backwards is most often used.

# MG-CFA: Partial Invariance (2)

- We can use modification indices to identify the constrained parameters we need to free.
- Once we have done this, our model is referred to as partially invariant.
- If we free a loading, we would also allow it's intercept and residual to be free.
- Interpretation of partially invariant models is much debated.

# MG-CFA: Partial Invariance (3)

```
# Configural Invariance
config <- cfa(model, data, group = "Sex")

# Metric
metric <- cfa(model, data, group = "Sex",
              group.equal=c("loadings"))

# Partial Metric
# Free the loading of N2
metric <- cfa(model, data, group = "Sex",
              group.equal=c("loadings"),
              group.partial=c("N=~N2"))
```

# Limits of `lavaan`

# Limits

- The biggest issue always used to be estimators.
  - A limited selection were available for more complex models.
  - This has improved significantly
    - I *think* `OpenMx` probably still has the edge in this respect.
    - Both lag on the performance of proprietary software such as MPlus.
- Not so much a limit, but the wrappers and related functions can mask issues.
  - This is not a `lavaan` issue, it is a general issue as complex models get easier to fit.
  - SEM is no exception to this.

That is all I have. Any questions?